Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Paulo André Leite Cerqueira

GBK-Pacer2 – Cadenciador para desporto individual de alta competição

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor Manuel Bernardo Salvador Cunha, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri

presidente

Professora Doutora Ana Maria Perfeito Tomé
Professora Associada do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor António Paulo Gomes Mendes Moreira
Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Manuel Bernardo Salvador Cunha
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Quero desde já agradecer aos meus orientadores, o Professor Doutor Manuel Bernardo Salvador Cunha e o Engenheiro Pedro Kulzer, pela oportunidade apresentada e pela confiança que depositaram em mim.
Um agradecimento especial também para todos os professores do Departamento de Electrónica, Telecomunicações e Informática que, de uma forma mais ou menos activa, contribuíram para a minha formação e incentivaram em mim um gosto especial pela área da electrónica.

Aos meus pais e irmãos, pelo incondicional apoio, motivação e por me proporcionarem todas as condições necessárias para concluir este desafio.

À minha namorada, por todo o carinho, motivação e especialmente compreensão. Por todos os momentos, acontecimentos e oportunidades. Por seres quem és… e como és…

À minha família e amigos mais próximos… e menos próximos.

palavras-chave

Cadenciador, Pacer, Treino, Natação, Atletismo, Ciclismo, Remo, Wireless, LED, Microcontrolador.

resumo

O projecto “GBK-Pacer2” consiste na elaboração de um cadenciador (pacer) para atletas de alta competição, tendo como base o melhoramento do cadenciador já existente GBK-Pacer, adaptado com novas tecnologias então desenvolvidas.

Este projecto foi iniciado há dois anos atrás. Foi já desenvolvido todo o software da interface gráfica que estará contido na PDA, e que inclui vários tipos de treino. Foi também desenvolvido um protótipo da pista de luzes, da buzina e de todos os restantes módulos, ficando todos eles a funcionar individualmente. Foi ainda iniciado, mas não concluído, o protocolo de comunicação a ser utilizado.

É esperado que após a conclusão deste trabalho, todo o código dos vários módulos, tenha sido revisto, comentado e optimizado, e que todo o sistema comunique correctamente. O software da PDA deverá ser adaptado para que esta consiga controlar todo o sistema.

Embora este projecto tenha sido desenvolvido numa fase inicial exclusivamente para auxiliar o treino competitivo de atletas de natação, a sua versatilidade e utilidade facilitam a expansão para vários tipos de treino de atletas, tais como o atletismo e o ciclismo de pista.

keywords

Cadence, Pacer, Training, Swimming, Athletics, Cycling, Rowing, Wireless, LED, Microcontroller.

abstract

The "GBK-Pacer 2" consists of developing a pacer equipment for elite athletes, based on improving the existing pacer called “GBK-Pacer”, while adapting to new technologies developed.

This project was started two years ago. Has already been developed all the graphical interface software that will be contained in the PDA, and includes various types of training. It has also been developed a prototype of the runway lights, horn and all the other modules, getting them all to work individually. Has also been started, but not completed, the communication protocol to be used.

It is expected that upon completion of this work, all the code of the various modules, has been revised, reviewed and optimized, and the entire system to communicate properly. The PDA software should be adapted so that it can control the whole system.

Although this project has been developed at an early stage only to assist the training of competitive swimmers, its versatility and utility enables its expansion to various types of training athletes, such as athletics and track cyclists.

Índice

Lista de Figuras

Lista de Tabelas e Gráficos

Lista de Acrónimos

ADC – **A**nalogic-to-**d**igital **C**onverter

ANACOM – **A**utoridade **N**acional de **Com**unicações

ASCII – **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

BSL – **B**oot**S**trap **L**oader

DAC – **D**igital-to-**a**nalogic **C**onverter

I/O – **I**nput/**O**utput

I2C – **I**nter-**i**ntegrated **C**ircuit

ID – **Id**entificação

ISM – **I**ndustrial, **S**cientific and **M**edical

JTAG – **J**oint **T**est **A**ction **G**roup

LCD – **L**iquid **C**rystal **D**isplay

LED – **L**ight **E**mitting **D**iode

LPM – **L**ow **P**ower **M**ode

MOSFET – **M**etal-**o**xide-**s**emiconductor **F**ield-**e**ffect **T**ransistor

NMI – **N**on-**m**askable **I**nterrupt

NRZ – **N**on-**r**eturn-to-**z**ero

PDA – **P**ersonal **D**igital **A**ssistant

PWM – **P**ulse-**w**idth **M**odulation

RDIS – **R**ede **D**igital **I**ntegrada de **S**erviços

RF – **R**adio **F**requency

RGB – **R**ed-**G**reen-**B**lue

RISC – **R**educed **I**nstruction **S**et **C**omputer

RSSI – **R**eceived **S**ignal **S**trength **I**ndication

RTC – **R**eal **T**ime **C**lock

SO – **S**istema **O**perativo

SPI – **S**erial **P**eripheral **I**nterface

SRD – **S**hort **R**ange **D**evice

UART – **U**niversal **A**synchronous **R**eceiver/**T**ransmitter

USART – **U**niversal **S**ynchronous/**A**synchronous **R**eceiver/**T**ransmitter

XOR – E**x**clusive **O**r

Capítulo 1. Introdução

1.1 - Contextualização

Em muitos desportos, tais como a natação, o atleta é incapaz de obter em tempo real um feedback da sua prestação para uma qualquer posição ao longo do seu treino [Garber, 2008]. Tradicionalmente o seu treinador apenas pode aconselhar o atleta acerca da sua performance no final de cada volta, pois durante grande parte do tempo a cabeça do nadador estará submersa debaixo de água o que dificultará a percepção de qualquer instrução que este lhe tente comunicar [Garber, 2008]. Além da dificuldade em escutar as instruções do seu treinador, o nadador não pode também contar que estas sejam passadas visualmente, pois a orientação da sua cabeça é muito importante na sua movimentação dentro de água e na fase de inspiração, tendo especial importância na minimização da força de atrito hidrodinâmico a que o nadador se sujeita, assim como favorecer a produção de força propulsiva pela acção dos segmentos motores [Marinho, 2003].

Para verificarmos as dificuldades de comunicação acima referidas, tomemos especial atenção à imersão dos ouvidos e ao campo visual muito limitado do nadador durante a execução da inspiração e da expiração nas quatro técnicas regularmente conhecidas: crol, costas, bruços e mariposa [Sacadura, 1988, p. 6]. Na *Figura 1.1* podemos verificar essas limitações que serão explicadas detalhadamente nos dois parágrafos seguintes.

A inspiração deverá ser efectuada frontalmente (nos estilos de bruços e mariposa), através da extensão da cabeça, ou lateralmente (no estilo de crol), através de uma rotação lateral da mesma

[Barbosa, 2003]. Deste modo, a cabeça manter-se-á completamente submersa na água com o olhar dirigido para o fundo da piscina durante todo o período de expiração [Sacadura, 1988, p. 76]. As inspirações laterais poderão ser do tipo unilateral ou bilateral [Barbosa, 2003]. Na inspiração unilateral, esta é efectuada através da rotação da cabeça sempre para um dos lados, enquanto na bilateral, a inspiração é efectuada através de uma rotação alternada para cada lado, em ciclos inspiratórios consecutivos [Barbosa, 2003]. Independentemente de esta ser unilateral ou bilateral, durante a sua execução um dos ouvidos estará submerso dentro de água. No período de expiração, ambos os ouvidos estarão submersos dentro de água devido à imersão de grande parte da cabeça e do olhar estar dirigido para o fundo da piscina [Marinho, 2003].

No estilo costas, o corpo deverá estar o mais horizontal possível, com a cabeça em posição natural no prolongamento do tronco [Marinho, 2003]. Neste caso, o olhar deverá estar dirigido para o tecto da piscina [Marinho, 2003], com a zona posterior da cabeça apoiada na água, ficando esta parcialmente em imersão (incluindo o aparelho auditivo) durante todo o período de execução deste estilo [Marinho, 2003].

Podemos assim verificar que, para maximizar o rendimento, o campo visual do atleta deverá estar reduzido ao fundo ou tecto da piscina durante as fases de expiração, e ao tecto, topo ou lateral da piscina durante as fases de inspiração, tornando o contacto visual com o treinador muito escasso e limitado. Podemos também verificar que o aparelho auditivo deverá encontrar-se parcial ou totalmente submerso como consequência da posição da cabeça, dificultando o reconhecimento e interpretação de comandos transmitidos verbalmente pelo treinador.

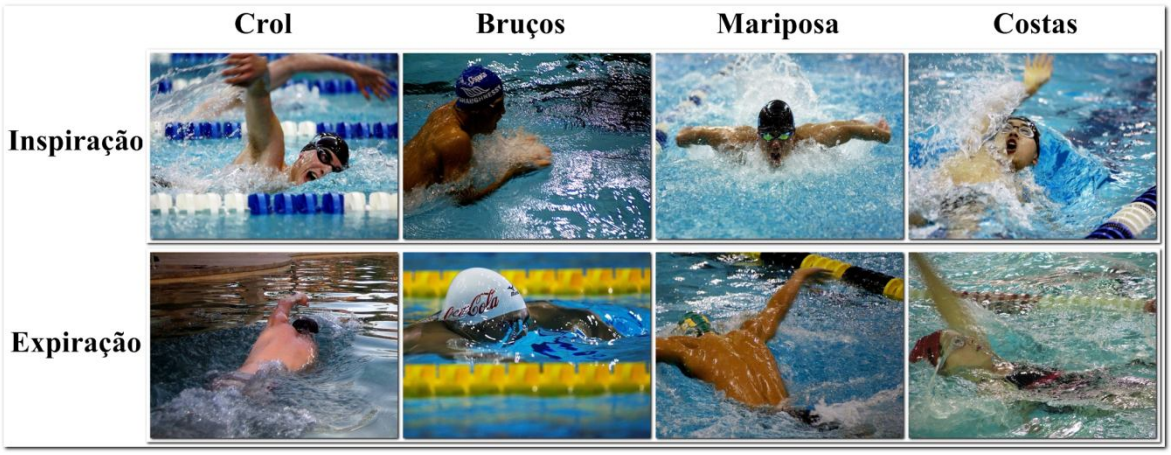


Figura 1.1- Inspiração e expiração para quatro diferentes estilos de natação[1]

Consequentemente o atleta apenas consegue ter uma visão geral de qual foi a sua performance ao fim de uma ou várias voltas, e não ao longo de várias secções em que uma piscina poderá ser divida, pois para analisar essa performance é necessário que o nadador interrompa o seu

[1] Imagens retiradas do *Flickr* em Outubro de 2009, com licenças de utilização. Autores das imagens: em cima (esquerda para a direita) - Angela Radulescu, Vironevaeh, Angela Radulescu e Angela Radulescu; em baixo (esquerda para a direita) – Pnoeric, Tpower1978, Vironevaeh e Virginia.

exercício [Garber, 2008]. Essa visão geral é-lhe transmitida pelo treinador com a apresentação dos tempos efectuados em cada volta, ou através da visualização da prova em gravações de vídeo [Garber, 2008].

Para contornar esta limitação, utilizam-se pistas luminosas subaquáticas de estimulação, denominadas de *pacers* (ou cadenciadores), que permitem ao atleta obter um treino mais eficiente, e alcançar assim melhores resultados [Termin, 1999]. O nadador recebe estímulos das luzes que tanto podem impor uma determinada frequência de braçadas conforme vão piscando, como uma determinada velocidade de natação imposta pelo acender sequencial das luzes, como ainda uma junção de ambos os estímulos [Termin, 1999]. De salientar que para o estilo de costas, a utilização de um *pacer* subaquático não terá qualquer vantagem pois o atleta nada de cara virada para o tecto da piscina.

1.2 - Estado da arte

Para o auxílio do treino de natação existem já vários *pacers* disponíveis no mercado.

Um dos primeiros *pacers* que utilizava luz para cadenciar o treino de natação de que temos conhecimento data do início dos anos noventa, mas infelizmente não foi possível reunir documentação acerca do mesmo, pois já há muito que não é utilizado. De acordo com informação recolhida junto do co-orientador da dissertação, Eng.º Pedro Kulzer, e em resultado da sua experiência pessoal, este era um pacer constituído por uma consola de programação que continha um *display LCD (Liquid Crystal Display)* cinzento de 2x16 caracteres, e por uma pista de 11 luzes (separadas de 2,5m) que eram colocadas no fundo da piscina (não tinha buzina embutida, era necessário que fosse o treinador a dar a partida). Cada luz era constituída por uma lâmpada incandescente e por um descodificador binário, que individualizava assim cada luz, embebidos em cola *epoxy* que os tornava estanques à água. Como as lâmpadas eram incandescentes, estas apresentavam um brilho muito fraco, fundiam muito facilmente e reduziam a autonomia do sistema drasticamente. A consola comunicava com a pista de luzes através de um cabo de 7 condutores, com 2 reservados para a alimentação (*VCC* e *GND*) e os restantes 5 para o endereçamento binário de cada luz[2]. No final da sua utilização, enrolava-se o cabo em volta da consola.

Em meados dos anos noventa, apareceu o cadenciador *GBK-Pacer* (da *GBK-Electronics*, entretanto extinta) que serviu de base para a elaboração deste projecto. O *GBK-Pacer* trata-se de um cadenciador para desportos de alta competição constituída por uma consola, uma buzina e até um máximo de 4 pistas de luzes. A consola de programação está conectada à buzina através de um cabo de 6 condutores, e a buzina comunica com a pista de luzes através de um cabo de 4 condutores (*VCC*, *GND*, *CLK* e *DATA/RESET*). O tipo de treino é pré-seleccionado na consola de um

[2] 2^5 = 32 combinações diferentes, que é um valor suficiente para endereçar as 26 luzes.

modo pouco *user-friendly*, sendo necessário a consulta do manual para efectuar a sua programação. Na *Figura 1.2* podemos ver uma imagem frontal e outra traseira da consola.

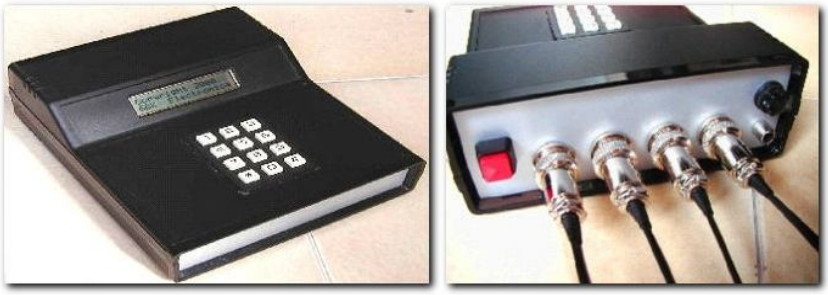


Figura 1.2 – Vista frontal e traseira da consola do GBK-Pacer[3]

A consola é constituída por um teclado numérico com tecla de asterisco e outra de cardinal e ainda por um monitor *LCD* de 2 linhas monocromático. A buzina é constituída por 1 sirene e por 6 *displays* de 7 segmentos. Cada pista de luzes é constituída por 11 ou 26 luzes, separadas de 2,5m ou 1m respectivamente, onde cada luz contém 3 *LED (Light Emitting Diode)* amarelos de alto brilho. Na *Figura 1.3* podemos ver os três *LED* utilizados em cada luz e o enrolador utilizado para armazenar os 25m de cabo de cada luz, assim como um pormenor dos *displays* de 7 segmentos da buzina.

Figura 1.3 - Pormenor da caixa de uma luz e respectivo enrolador e pormenor dos displays da buzina[4]

Todo este sistema funciona a baixa tensão e é por isso alimentado a baterias, apresentando uma autonomia de 5h a 12h em funcionamento contínuo, dependendo do número de pistas e do número de luzes ligadas. O tempo de carga das baterias é de aproximadamente 15h e tem um tempo de vida estimado de 500 cargas completas.

Algumas das desvantagens deste sistema são: electrónica maioritariamente analógica, consola de grandes dimensões quando comparado com o existente hoje em dia, requer um elevado número de cabos para interligar todo o sistema, elevando drasticamente o seu peso, e finalmente a sua já referida interface muito pouco *user-friendly* [Cabrita, 2007].

Outro *pacer* relevante, mas muito mais recente é o *pacer* patenteado em 2007 por um sul-africano. Este *pacer* apresenta algumas semelhanças com o *GBK-Pacer* e/ou com o *GBK-Pacer2*. É

[3] Imagem retirada da *Web* em Novembro de 2009. Endereço: http://www.kulzertec.com/Pacer1.html.
[4] Imagem retirada da *Web* em Novembro de 2009. Endereço: http://www.kulzertec.com/Pacer1.html.

colocado também ele no fundo da piscina; é constituído por uma cadeia de vários *LED* consecutivos, controlados individualmente por um microcontrolador dedicado; utiliza uma estação de controlo programável para seleccionar os diversos tipos de treino e suas durações; a estação contém um pequeno *display* onde mostra a velocidade e distância percorrida pelo nadador; e finalmente, a estação de controlo comunica com cada luz através de um cabo.

Ao contrário do *GBK-Pacer*, este utiliza um transmissor posicional transportado pelo atleta que comunica via *RF (Radio Frequency)* com a consola, enviando a posição relativa actual do nadador, para que a consola calcule a velocidade instantânea aproximada do atleta.

Na *Figura 1.4* podemos visualizar a instalação do sistema no fundo de uma piscina. Cada secção da pista de luzes corresponde a um *LED* e estes encontram-se equidistantes entre eles. No topo da piscina está a consola programável que controla todos os *LED* e que comunica com estes através de um cabo. Como o sistema estará colocado no fundo da piscina, a estanquicidade é obrigatoriamente indispensável para que tudo funcione. Podemos ainda visualizar um esboço da consola com mais pormenor, onde são visíveis os botões necessários para a sua programação, o *display* que mostrará informação ao treinador e ao atleta e a antena que receberá os dados do transmissor posicional transportado pelo atleta [Garber, 2008].

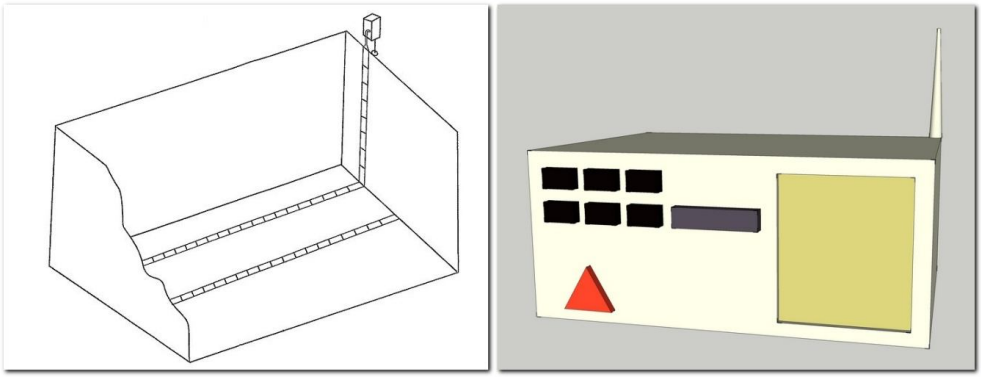


***Figura 1.4 - Sistema colocado no fundo da piscina com a consola fora desta e consola programável*[5]**

Finalmente, também não podemos deixar de mencionar um pacer patenteado em 1985 por um norte-americano que consiste num painel a ser colocado na borda da piscina, no final da pista. Este *pacer* tem três mostradores, um com o tempo total, outro com o tempo da volta e outro com o número de voltas. Este *pacer* contem um sensor que sempre que pressionado pelo atleta no final de cada volta, apresenta o tempo da volta e incrementa o número de voltas efectuadas pelo atleta. Embora seja um *pacer* que tem muito pouco de relação com o *GBK-Pacer*, é um cadenciador que se distingue da maioria dos cadenciadores por ser activado e controlado exclusivamente pelo atleta, mostrando ao seu treinador apenas os tempos e dados mais relevantes. Na *Figura 1.5* podemos visualizar este *pacer*, colocado no topo da piscina com os três mostradores e com o sensor de pressão submerso dentro de água [Dawley, 1985].

[5] Imagem da esquerda retirada de: [Garber, 2008]; imagem da direita elaborada segundo uma imagem do mesmo documento.

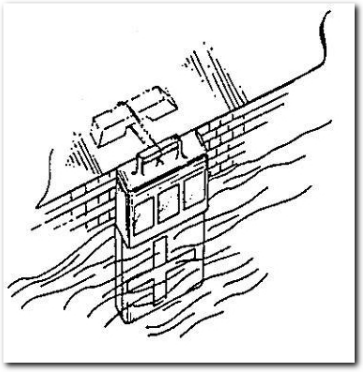


Figura 1.5 - Colocação do pacer no topo da piscina[6]

1.3 - Objectivos

Este projecto consiste em continuar o trabalho já efectuado em anos anteriores por dois alunos da Universidade de Aveiro, na área de projecto de final de curso, para o *GBK-Pacer2*.

Um deles teve como objectivo desenvolver o *hardware* e *software* da pista de luzes e de todos os módulos necessários para o seu controlo através de uma *PDA (Personal Digital Assistant)*. Este projecto, quando concluído, conseguiu que todos os módulos funcionassem individualmente, faltando ainda implementar a comunicação *wireless* [Cabrita, 2007].

O segundo teve como objectivo desenvolver o *software* de interface gráfica da *PDA*, implementado numa plataforma portátil *Pocket PC,* equipado com o sistema operativo *Microsoft Windows Mobile*. Este projecto, quando concluído, conseguiu desenvolver uma boa interface gráfica, mas ficou por concluir a comunicação com o *software* e *hardware* do primeiro aluno [Silva, 2007].

Para nos auxiliar neste projecto, teremos como referências os *datasheets* dos componentes utilizados e os relatórios finais de projecto dos dois alunos.

Os principais objectivos deste trabalho podem ser assim divididos em três tarefas distintas e bem definidas.

A primeira tarefa consiste em analisar e compreender todo o *software* e *hardware* já existente para todos os módulos, à excepção do *software* da interface gráfica da *PDA*.

A segunda tarefa consiste em completar o *software* e *hardware* com as funcionalidades que ainda não estão implementadas, como por exemplo a comunicação *wireless*. Todas essas funcionalidades terão que ser convenientemente testadas e no final todos os módulos terão que ser capazes de comunicar correctamente entre si. Será ainda necessário optimizar, organizar e comentar correctamente todo o código, sendo imprescindível que o programa das luzes seja

[6] Imagem retirada de: [Dawley, 1985]

reduzido de modo a ocupar no máximo *1kB* para caber na memória de um microcontrolador *MSP430F1101A*. Será também necessário fazer os testes convenientes de alcance e consumo aos módulos *wireless* para concluir se serão ou não os indicados para este projecto. No final teremos que realizar os esquemáticos e os manuais que se achem necessários para uma boa referência futura.

No final destas duas tarefas é imprescindível que todo o sistema esteja a funcionar e a comunicar correctamente e que todos os testes de alcance e consumo tenham sido realizados para se poder avançar para a tarefa seguinte. Estas duas tarefas incidem exclusivamente sobre o trabalho começado pelo aluno responsável pelo desenvolvimento do *software* e *hardware* da pista de luzes e de todos os módulos necessários para o seu controlo através de uma *PDA*.

Finalmente, a terceira tarefa consiste em analisar todo o *software* da *PDA*, colocando esta a iniciar e a controlar todo o sistema, e elaborar um novo protótipo que será muito semelhante ao produto final. Detectar possíveis falhas que tinham até então passado despercebidas e estudar a melhor forma de colocação do sistema no fundo da piscina, da sua impermeabilidade e protecção contra pirataria, assim como a possibilidade de adaptação do sistema para a prática de outras modalidades. Esta última tarefa incide inicialmente sobre o trabalho realizado pelo segundo aluno a fim de desenvolver um protocolo de comunicação entre a *PDA* e os restantes módulos.

1.4 - Meios de suporte ao projecto

A estrutura do *Pacer2* pode ser definida como apresentado na *Figura 1.6*. Nessa figura, podemos verificar que a estrutura base do *Pacer2* consiste em dois blocos principais, sendo eles os blocos da *PDA* e da piscina. Estes dois blocos comunicarão entre si via *wireless*, e todos os módulos e componentes existentes dentro desses dois blocos comunicarão entre si através de ligações físicas.

O bloco da *PDA* é constituído pelo *software* criado propositadamente para o *Pacer2* a correr sobre o *SO (*Sistema Operativo*)* nativo da *PDA* e por um módulo *wireless* que é responsável por enviar e receber dados via *RF*.
O módulo *wireless* é composto por um *MAX232* que permite a comunicação por *RS232* com a *PDA*, por um *transceiver* que permite enviar e receber dados por *RF* e um microcontrolador que processa os dados enviados/recebidos por *RF* e os dados enviados/recebidos pela *PDA*.

O bloco da piscina é constituído por quatro outros módulos que correspondem ao módulo *wireless*, à buzina, ao reprogramador da buzina e à pista de luzes. O módulo *wireless* será composto apenas por um *transceiver* que permitirá enviar/receber dados por *RF*.

O módulo da buzina será composto por um microcontrolador que programará o módulo *wireless*, processará os dados enviados/recebidos por esse mesmo módulo e os dados enviados/recebidos pela pista de luzes. Conterá ainda um *LCD* para disponibilizar várias informações ao atleta/treinador e uma sirene para sinalizar a partida.

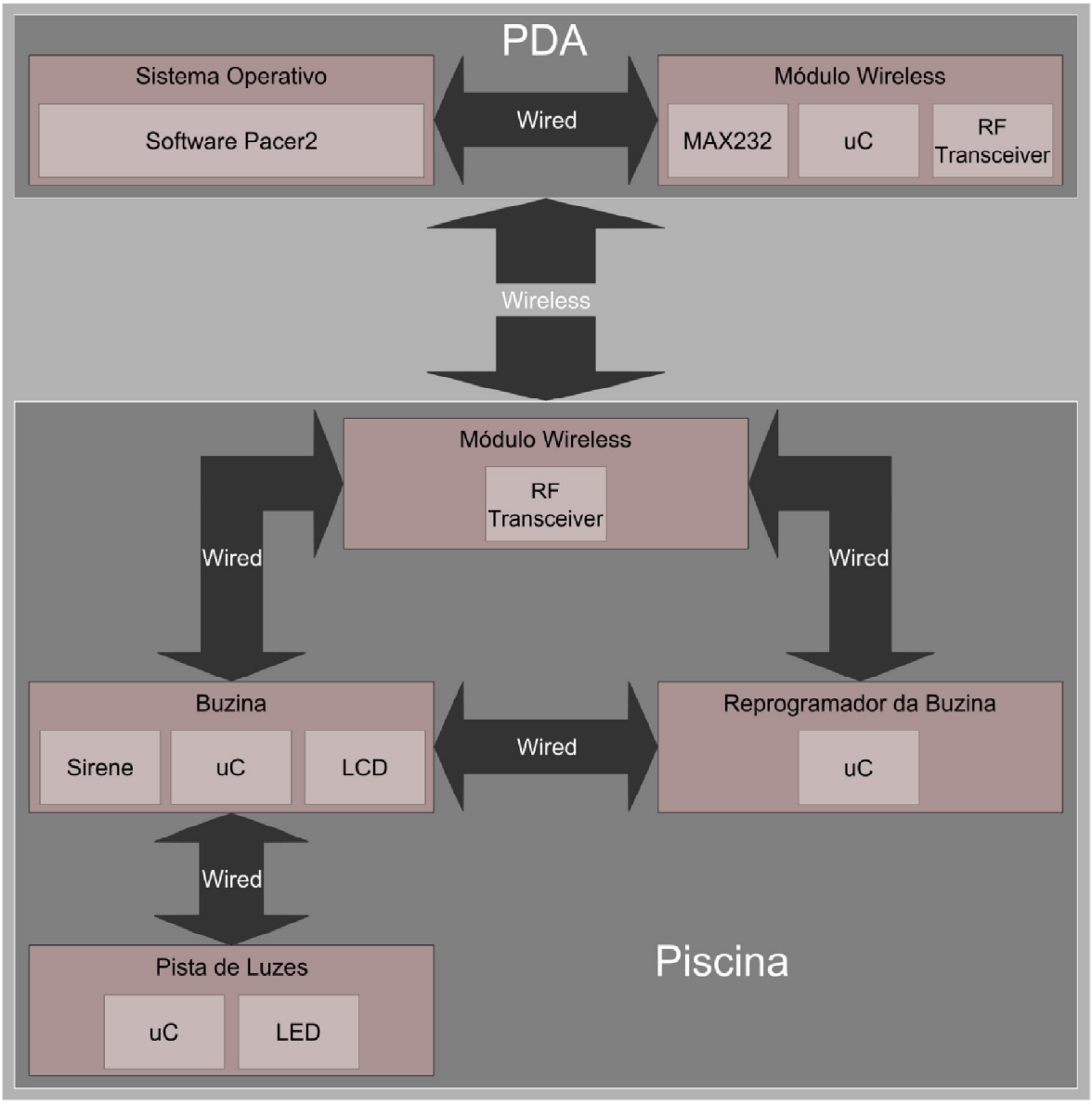


Figura 1.6 - Estrutura do Pacer2

O módulo do reprogramador da buzina será composto exclusivamente por um microcontrolador que configurará o módulo *wireless*, processará os dados enviados/recebidos por esse mesmo módulo e reprogramará via *BSL* (*Bootstrap Loader*) o microcontrolador da buzina.

O módulo da pista de luzes é composto por 26 luzes contendo cada uma delas um microcontrolador e um *LED RGB (Red-Green-Blue)*. O microcontrolador associado a cada luz terá associado um *ID (IDentificação)* único que permite a sua individualização perante as restantes. Assim, cada microcontrolador processa os comandos recebidos pela buzina e só os executa se determinar que os comandos têm como destino o seu *ID*. De realçar que sempre que falarmos no termo luz, referimo-nos a um conjunto de um microcontrolador com o seu respectivo *LED* e sempre que falarmos em pista de luzes, referimo-nos ao módulo constituído pelo conjunto de 26 luzes.

Uma descrição mais detalhada de cada componente será fornecida mais à frente neste capítulo.

Com o cadenciador *Pacer2*, o atleta conseguirá assim obter uma resposta em tempo real da sua performance para cada uma das várias secções da piscina e para cada uma das diferentes fases do seu treino. A piscina passa a estar dividida em 25 secções de 1 metro terminadas por um *LED RGB* de alto brilho, num total de 26 *LED* (25 *LED* para cada metro da piscina e um adicional para a posição 0) em que a cor de cada *LED* pode representar uma instrução do seu treinador e a cadência com que estes acendem e/ou apagam marcam um ritmo e/ou velocidade facilmente perceptível. Para piscinas de 50 metros será necessário utilizar duas buzinas com duas pistas de luzes, colocadas em cada topo da piscina.

O sinal de partida, juntamente com todos os comandos necessários para a realização do treino, será enviado via *wireless* por uma *PDA,* controlada pelo treinador, e recebida pela buzina que estará instalada no topo da piscina. Os comandos serão recebidos pelo módulo *wireless* da piscina e tratados depois pelo microcontrolador da buzina, de forma a activar a sirene e sinalizar a partida do atleta. Os restantes comandos serão enviados para a pista de luzes através de um cabo de 4 fios que conectará todo o sistema de luzes à buzina.

De modo a tornar possível futuros *firmware updates*, tanto a buzina como as luzes foram desenvolvidas para que possam ser fácil e rapidamente reprogramadas via *BSL*. No caso da pista de luzes será o próprio microcontrolador da buzina quem inicializará simultaneamente o *BSL* em todos os microcontroladores das luzes. Este processo é efectuado sempre que se liga a buzina e o programa das luzes estará contido num bloco de memória dentro do programa da buzina. O programa da buzina estará por sua vez contido dentro da *PDA*, no formato de um ficheiro. Sempre que existir um novo *firmware update* (tanto para a pista de luzes, como para a buzina), basta substituir esse ficheiro pelo mais recente para que todo o *firmware* seja actualizado automaticamente. No caso da buzina, como este novo *firmware* terá que ser recebido via *wireless*, é necessário a existência de um microcontrolador adicional capaz de configurar o módulo *wireless* para receber esse novo *firmware* e para inicializar o *BSL* no microcontrolador da buzina. Este microcontrolador adicional é o denominado de reprogramador da buzina.

O facto de as luzes serem reprogramadas sempre que se ligue a buzina, tem como objectivo tornar o sistema de *firmware updates* mais *user-friendly* para que este seja acessível e compreensível para todo o tipo de pessoas, mesmo para aquelas que não tenham grande experiência em informática ou electrónica. Consegue-se assim garantir um correcto *firmware update* através da simples substituição de um único ficheiro.

1.4.1 - Componentes

Serão apresentados de seguida os componentes seleccionados para cada um dos diferentes módulos.

1.4.1.1 - Módulo Wireless

Optou-se por testar um módulo pré-fabricado *MMCC1000*[7] da *Propox* configurado para uma utilização a 433*MHz* e um outro configurado para uma utilização a 868*MHz*. Existem no entanto algumas limitações impostas pela *ANACOM (Autoridade NAcional de COMunicações)* em Portugal à utilização destas duas gamas de frequências, que consistem no seguinte[8]:

- A 433*MHz* pode-se utilizar uma potência de transmissão de até 10*mW*, desde que se garanta um *duty cycle*[9] inferior a 10%;
- A 433*MHz* pode-se utilizar um *duty cycle* até 100%, desde que se garanta que a potência de transmissão não seja superior a 1*mW*;
- A 868*MHz* pode-se utilizar uma potência de transmissão até 25*mW*, desde que se garanta um *duty cycle* inferior a 1%.

Na *Figura 1.7* podemos ver um desses módulos produzidos pela *Propox*. Optou-se pela utilização deste módulo pré-fabricado, pois já vem pronto a funcionar como receptor ou transmissor, poupando-se assim o trabalho de escolher as resistências e condensadores adequados para configurar o seu funcionamento numa determinada frequência. Estes módulos utilizam o *transceiver RF CC1000*[10] da *TI*.

***Figura 1.7 - Módulo MMCC1000*[11]**

A *Figura 1.8* mostra o esquema do circuito de configuração do módulo, em que as resistências, condensadores, bobinas e cristal têm o valor referido na *Tabela 1.1 e na Tabela 1.2* consoante a frequência e o tipo de encapsulamento que se pretenda utilizar.

---

[7] [Propox, 2009]
[8] [Anacom, 2008]
[9] Segundo a definição da *ANACOM*, é a relação, expressa em percentagem, do tempo máximo em que um equipamento se encontra activo com uma ou mais portadoras, relativamente a um período de uma hora. [Anacom, 2008]
[10] [Texas Instruments, 2009]
[11] Imagem retirada da *Web* em Julho de 2009: http://www.propox.com/products/t_92.html.

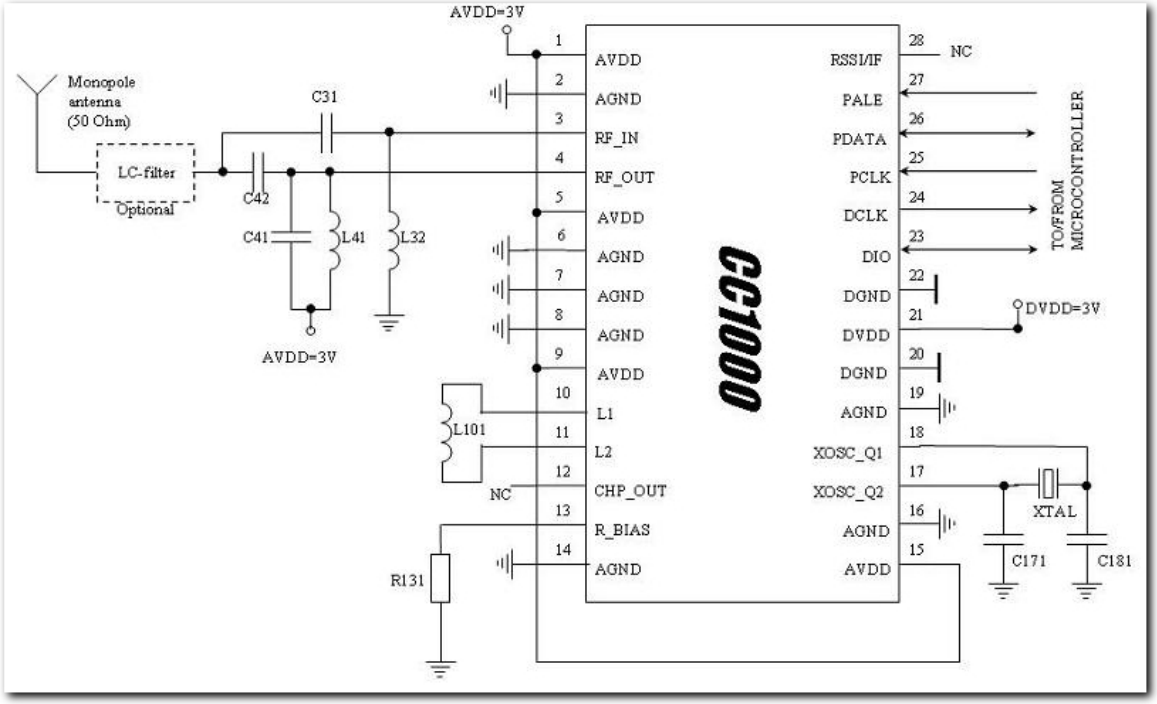


Figura 1.8 - Esquema CC1000[12]

CC1000 TSSOP package				
Item	315MHz	433MHz	868MHz	915MHz
C31	8.2pF	15pF	10pF	10pF
C41	2.2pF	8.2pF	Not Used	Not Used
C42	5.6pF	5.6pF	4.7pF	4.7pF
C171	18pF	18pF	18pF	18pF
C181	18pF	18pF	18pF	18pF
L32	39nH	68nH	120nH	120nH
L41	20nH	6.2nH	2.5nH	2.5nH
L101	56nH	33nH	4.7nH	4.7nH
R131	82kΩ	82kΩ	82kΩ	82kΩ
XTAL	14.7456MHz, 16pF load	14.7456MHz, 16pF load	14.7456MHz, 16pF load	14.7456MHz, 16pF load

Tabela 1.1 - Lista de Componentes de configuração do CC1000 para diferentes frequências (TSSOP)[13]

[12] Imagem retirada de: [Texas Instruments, 2009]
[13] Tabela retirada de: [Texas Instruments, 2009]

CC1000 UltraCSP package				
Item	315MHz	433MHz	868MHz	915MHz
C31	8.2pF	15pF	10pF	10pF
C41	Not Used	Not Used	Not Used	Not Used
C42	4.7pF	4.7pF	6.8pF	6.8pF
C171	18pF	18pF	18pF	18pF
C181	18pF	18pF	18pF	18pF
L32	39nH	68nH	120nH	120nH
L41	22nH	15nH	2.7nH	2.7nH
L101	56nH	33nH	7.5nH	7.5nH
R131	82kΩ	82kΩ	82kΩ	82kΩ
XTAL	14.7456MHz, 16pF load	14.7456MHz, 16pF load	14.7456MHz, 16pF load	14.7456MHz, 16pF load

Tabela 1.2 - Lista de Componentes de configuração do CC1000 para diferentes frequências (UltraCSP)[14]

Este módulo traz incorporado um *chip CC1000* emissor e receptor sem fios na banda *UHF* de muito baixo consumo energético e funcionamento a baixa voltagem. No seu *datasheet* tem todas as informações necessárias para a projecção de um circuito para operar em qualquer frequência entre a gama 300-1000*MHz*. A *Tabela 1.1* e a *Tabela 1.2* contêm os valores dos componentes a alterar no circuito da *Figura 1.8*, para cada gama de frequências de utilização.

O circuito é sobretudo projectado para operar nas bandas de frequências *ISM (Industrial, Scientific and Medical)* e *SRD (Short Range Device)* a 315, 433, 868 e 915*MHz*. Os 22 registos internos do *CC1000* podem ser programados através de uma interface série e com a ajuda do programa *SmartRF Studio* e do respectivo *datasheet*, tornando-o assim uma escolha bastante flexível e fácil de configurar.

Estes registos permitem a configuração de vários parâmetros, entre eles: frequência de operação[15], frequência do cristal interno ou externo, modo receptor ou transmissor, taxa de transmissão[16], modo de recepção/transmissão de dados[17], potência de transmissão[18], activar ou desactivar *RSSI (Received Signal Strength Indication)*, etc.

[14] Tabela retirada de: [Texas Instruments, 2009]

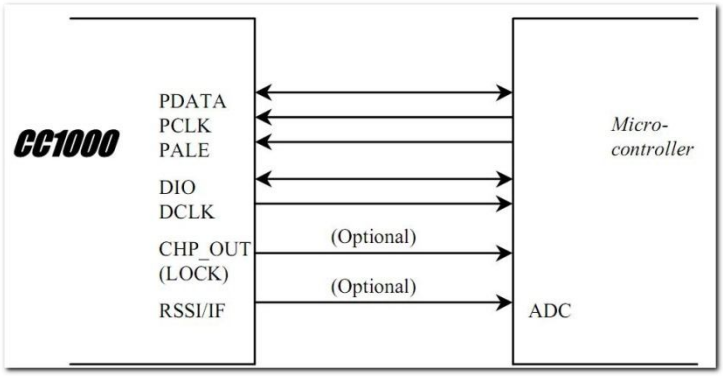
[15] Para alterar a frequência de operação, pode ser necessário alterar também o valor de alguns componentes passivos. Consulte a *Tabela 1.1* ou a *Tabela 1.2* para confirmar os componentes.

[16] A taxa de transmissão é medida em *baud rate*.

[17] O modo de recepção/transmissão refere-se ao modo em como o *chip* troca dados com o microcontrolador e para este *chip* em particular podem ser: modo *NRZ* síncrono, modo de codificação *Manchester* síncrono ou modo *UART* assíncrono transparente. Consulte o manual para uma melhor definição de cada modo.

[18] A potência de transmissão pode ser configurada entre -20 e 10*dBm*.

É usado tipicamente em conjunto com um microcontrolador e alguns componentes externos passivos como mostrado na Figura 1.9.



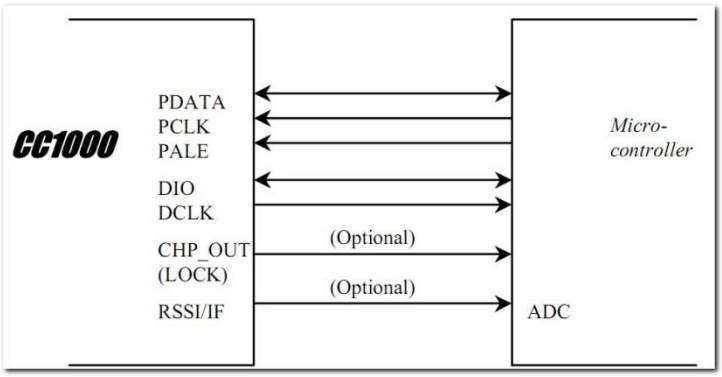


Figura 1.9 - Interface de comunicação com o microcontrolador[19]

Este módulo foi o único disponibilizado para a realização deste projecto, mas outras soluções, tal como o uso de módulos XBee, ZigBee ou Bluetooth, deverão ser estudadas em trabalho futuro. O seu custo *versus* vantagens/desvantagens deve ser tomado em especial atenção.

1.4.1.2 - Microcontrolador

Optou-se pela utilização de microcontroladores *MSP430 Ultra-Low Power Microcontrollers* da *TI*, pois são microcontroladores de baixo consumo energético, garantindo assim uma maior autonomia do sistema, funcionam a tensões muito baixas (1.8 a 3.6*V*), utiliza uma arquitectura *RISC (Reduced Instruction Set Computer)* de 16*bits*, a empresa disponibiliza amostras gratuitas, são relativamente baratos e, principalmente, porque permitem três formas de programação com poucos fios, *BSL* (2 fios), *JTAG* (*Joint Test Action Group*) - pelo menos 5 fios - e *Spy-by-Wire* (2 fios), permitindo a possibilidade de firmware updates através do uso de apenas 4[20] fios (*BSL* ou *Spy-by-Wire*).

[19] Imagem retirada de: [Texas Instruments, 2009]
[20] Sendo dois fios usados para a alimentação (*VCC* e *GND*) e outros dois fios para o modo utilizado.

Para o módulo da buzina foi escolhido o *MSP430F1611*[21], um microcontrolador com 5 diferentes modos de funcionamento, uma *ADC (Analogic-to-Digital Converter)* de 12*bits*, duas *DAC (Digital-to-Analogic Converter)* de 12*bits*, dois *timers* de 16*bits*, um comparador, duas interfaces de comunicação série *USART (Universal Synchronous/Asynchronous Receiver/Transmitter)* que podem ser usadas em modo *UART (Universal Asynchronous Receiver/Transmitter)* assíncrono, *SPI (Serial Peripheral Interface)* síncrono ou *I2C* (*Inter-Integrated Circuit*) - apenas na *USART0* - e com 10*kB* de memória *RAM* e 48*kB*+256*B* de memória *flash*. De realçar que a buzina terá que ter bastante memória, pois terá que guardar nessa memória o seu programa, assim como o programa das luzes que será enviado para estas por *BSL* e terá ainda que ter bastante memória livre para suportar futuros *firmware updates*.

Para o módulo das luzes, como o programa destas será significativamente mais reduzido (à volta de 1*kB*) e como não serão necessárias algumas funcionalidades do *MSP430F1611*, tais como a *ADC* ou o módulo *USART*, optou-se pela utilização do *MSP430F1101A*[22] com 1*kB*+128*B* de memória *flash* e 128*B* de memória *RAM*, 3 modos diferentes de funcionamento (*Active mode*, *Standby mode* e *Off mode*), um *timer* e um comparador. De referir também que como este microcontrolador não possui módulo *USART*, todas as funções necessárias à comunicação com terceiros deverão utilizar interrupções. A utilização deste microcontrolador foi decidida exclusivamente por razões de custo. Como cada pista de luzes contem 26 microcontroladores, é crucial escolher o mais barato pois uma diferença de custo, por mais pequena que seja, será sempre multiplicada por um factor de 26 para a construção de uma única pista de luzes.

Para o módulo do reprogramador da buzina e para módulo do simulador da *PDA* não está ainda definido qual será o microcontrolador a utilizar. Apesar de ambos utilizarem interrupções em detrimento do protocolo *USART* nas comunicações, estes necessitarão de uma memória bem maior do que o 1*kB* disponível no *MSP430F1101A*. A solução poderá passar pelo uso de microcontroladores da mesma família que o mencionado, mas com maior memória *RAM*, suficiente para alojar a versão final dos respectivos programas e deixar ainda alguma memória livre que poderá ser utilizada em futuros *firmware updates*.

No desenvolvimento deste projecto, serão no entanto, utilizados sempre os microcontroladores *MSP430F1611* pois são implementados em módulos de desenvolvimento, mais concretamente o *MSP430-H1611 HEADER BOX* da *OLIMEX*, que apresentam uma facilidade de programação e substituição muito grandes, condições essências para o desenvolvimento deste tipo de projectos.

Para a programação destes módulos será utilizado o *software IAR Embedded Workbench for MSP430*, pois é uma ferramenta para construção e *debugging* de aplicações *embedded*, apresenta uma interface com disponibilização do código em *Assembly*, compilador de linguagem *C/C++* e editor de texto. O *debugging* é efectuado através de *JTAG*.

[21] [Texas Instruments, 2002]
[22] [Texas Instruments, 1999]

1.4.1.3 - *Display LCD*

O *display* utilizado é o modelo *LCD EA DIP204B-6NLW* da *Electronic Assembly*. Trata-se de um *display* de 4*x*20 caracteres brancos sobre fundo azul, data bus de 8*bits*, modo *SPI*, 16 ícones estáticos na parte superior (nível da bateria, disquete, envelope, telefone, sino, setas, etc), 240 caracteres pré-definidos, 8 caracteres personalizáveis, 18 comandos (leitura, escrita, controlo do cursor, *scroll,* etc). Além de tudo isto, o *display LCD* vem já montado e encapsulado, pronto a ser soldado no circuito da buzina. Na *Figura 1.10* temos uma imagem do *display LCD*.

***Figura 1.10 - Display LCD*[23]**

1.4.1.4 - *LED*

OS *LED* utilizados para a pista de luzes são os *RGB LED Superflux 4-Pin* da *LED1.de*. São *LED* relativamente baratos, multicolores *RGB*, com um ângulo de visão de 100°, baixa tensão, aguenta picos de corrente superiores a 20*mA* por cor, garantindo assim o controle da intensidade do seu brilho por *PWM (Pulse-Width Modulation)*. Na *Figura 1.11* vemos uma imagem deste *LED* e os seus respectivos pinos onde verificamos que este funciona com lógica negativa, quer isto dizer que para ligar uma determinada cor no *LED* é necessário que o pino correspondente a essa cor esteja ligado a *GND* (o ânodo está ligado a VCC e o os três pinos correspondentes às três cores *RGB* são cátodos).

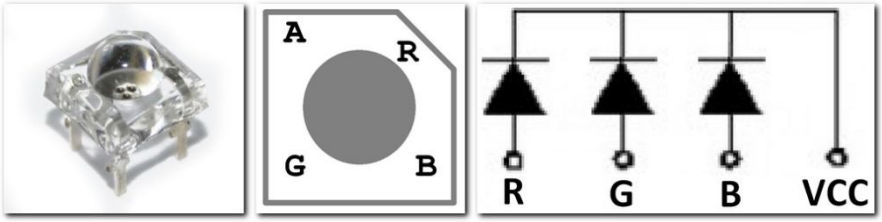


***Figura 1.11 - LED RGB e respectiva pinagem*[24]**

---

[23] Imagem retirada da *Web* em Outubro de 2009. Endereço: http://pt.mouser.com/search/include/LargeProductImage.aspx?path=electronicassembly/lrg/dip204b-6nlw.jpg.

[24] Imagem à esquerda retirada da *Web* em Outubro de 2009. Endereço: http://www.led1.de/shop/popup_image.php?pID=399&image=0&xploidID=62fb40c30bdde46be5f4c97762508d28.
Restantes imagens baseadas na informação disponível no seguinte docomento: www.led-tech.de/en/**Superflux**-LEDs_DB-7.pdf.

1.4.1.5 - Outros

A buzina comunicará com a pista de luzes através de um cabo telefónico *RDIS (*Rede Digital Integrada de Serviços*)* de 4 condutores multifilares. A escolha do cabo prendeu-se apenas ao seu preço, sendo que as características de cada cabo não variam muito entre si. Sendo assim, optamos pelo mais barato.

Para alimentar o sistema, foi escolhido o regulador de tensão *LD1117AV33* da *SGS-Thomson Microelectronics* que coloca uma tensão de saída constante de 3.3*V* quando na entrada recebe uma tensão até 15*V*. Este regulador de tensão tem protecção contra curto-circuito e um limitador interno de corrente capaz de fornecer até um máximo de 800*mA*.O esquema de ligação do regulador de tensão pode ser visualizado na *Figura 1.12*.

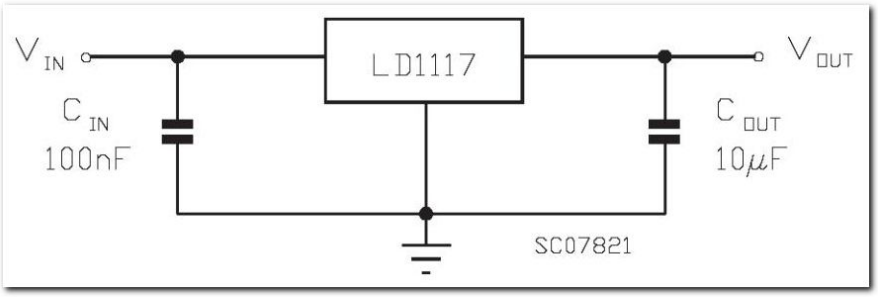


Figura 1.12 - Esquema de ligação do LD1117[25]

Tal como em todos os circuitos, serão também utilizados vários componentes passivos, tais como resistências ou condensadores, que não iremos aqui especificar.

1.4.2 - *BootStrap Loader*

O *BSL* permite ao utilizador do *MSP430* comunicar com a memória durante a fase de desenvolvimento, fabrico ou utilização do sistema projectado. Permite a escrita e a leitura de ambas as memórias, *RAM* e *flash*. O *BSL* utiliza um protocolo *UART/RS232* o que torna bastante flexível a sua utilização, tanto em *hardware* como em *software*.

Para entrar no *BSL* é necessária uma sequência de entrada que deve ser aplicada em alguns pinos específicos do *MSP430*, uma sequência de comandos iniciará a função desejada. Uma sessão de *"bootloading"* pode ser terminada continuando a operação num determinado *program address* ou aplicando um simples *reset* ao dispositivo. O acesso à memória via *BSL* é protegido contra utilização errada através de uma palavra-chave definida pelo utilizador.

[25] Imagem retirada de: [STMicroelectronics, 2009]

1.4.2.1 - Sequência de entrada em *BSL*

Para entrar em *BSL* num dispositivo com pinos *JTAG* partilhados (como é exemplo o microcontrolador *MSP430F1101A*) inicialmente devem estar os pinos *RST* e *Test* ambos a *low*, de seguida deve ocorrer pelo menos um impulso no pino *Test* e, por último, aquando de um novo impulso no pino *Test*, enquanto este ainda está a *high*, o pino *RST* deve passar a estar também ele a *high*. Nesta altura o microcontrolador entra em modo *BSL*. A *Figura 1.13* mostra a sequência de entrada em *BSL* para dispositivos com pinos *JTAG* partilhados.

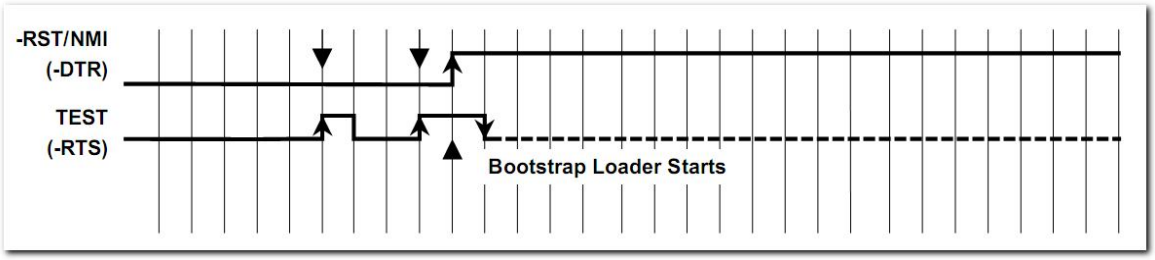


Figura 1.13 - Sequência de entrada em modo BSL para dispositivos com pinos JTAG partilhados[26]

Para entrar em *BSL* num dispositivo com pinos *JTAG* dedicados (como é o exemplo o microcontrolador *MSP430F1611*) o processo é, em tudo, semelhante ao anterior, substituindo-se apenas o pino *Test* pelo pino *TCK* e invertendo o estado deste sinal. A *Figura 1.14* mostra a sequência de entrada em *BSL* para dispositivos com pinos *JTAG* dedicados.

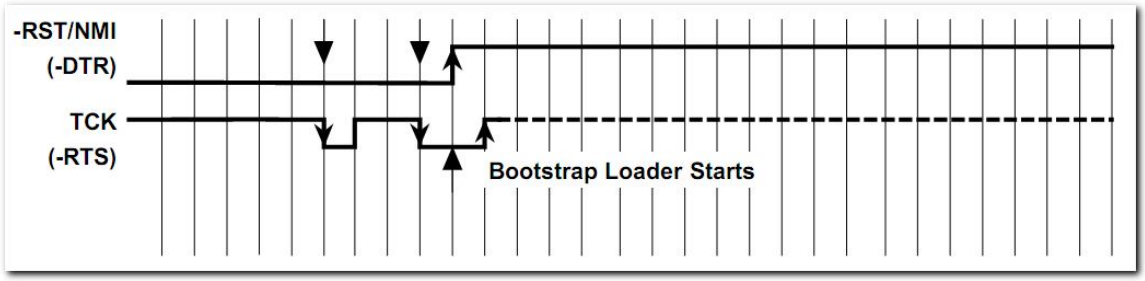


Figura 1.14 - Sequência de entrada em modo BSL para dispositivos com pinos JTAG dedicados[27]

Como ao longo do desenvolvimento do projecto foram utilizados dispositivos com pinos *JTAG* dedicados, em todas as figuras e diagramas que aqui apresentaremos será sempre representado o pino *TCK*.

1.4.2.2 - Protocolo de comunicação do *BSL*

O *BSL* comunica através de *UART* com as seguintes especificações:

[26] Imagem retirada de: [Texas Instruments, 2001]
[27] Imagem retirada de: [Texas Instruments, 2001]

- *Baud rate* de 9600*baud* em modo *half-duplex*
- 1 *Start bit*, 8*bits* de dados, 1 *Parity bit* e 1 *Stop bit*
- O pino *P1.1* é utilizado como pino de transmissão (*TX*)
- O pino *P2.2* é utilizado como pino de recepção (*RX*)

1.4.2.3 - Comandos do *BSL*

Antes de ser enviado qualquer comando é necessário enviar um *byte* de sincronização (*SYNC*). O valor desse *byte* é 0*x*80. Quando recebido este *byte* de sincronização o *BSL* responde com o envio do *byte DATA_ACK,* que tem o valor 0*x*90 e que serve para confirmar a recepção correcta do *byte* de sincronização.

Existem dois tipos de comandos no *BSL*, comandos desprotegidos: *Receive password*, *Mass erase*, *Transmit BSL version* e *Change baud rate*; e comandos protegidos por palavra-chave: *Receive data block*, *Transmit data block*, *Erase segment*, *Erase check*, *Load program counter* e *Start user program*.

Para executar os comandos protegidos por palavra-chave é necessário desbloquear primeiro o acesso a esses comandos enviando a palavra-chave correcta. Uma vez recebida essa palavra-chave e desbloqueado assim o acesso aos comandos, estes permanecem sempre acessíveis até que se execute uma sequência de *reset*.

O envio de comandos para o *BSL* terá que respeitar uma estrutura/sequência de *bytes*, sendo que alguns deles são obrigatórios e outros opcionais. Os 8 primeiros *bytes* (*HDR*, *CMD, L1*, *L2*, *AL*, *AH*, *LL* e *LH*) são obrigatórios, os *bytes D1* a *Dn* são opcionais, os *bytes CKL* e *CKH* são também eles obrigatórios e o *byte ACK* também é obrigatório, excepto para o comando *Transmit data block*. A *Figura 1.15* mostra o *data frame* dos comandos do *BSL*.

HDR	CMD	L1	L2	AL	AH	LL	LH	D1…Dn	CKL	CKH	ACK

Figura 1.15 - Data frame dos comandos do BSL[28]

- O *byte HDR* é o *byte* de *header* e é sempre 0x80;
- O *byte CMD* identifica o comando a ser executado;
- Os *bytes L1* e *L2* indicam o número de *bytes* que fazem parte do *data frame* desde *HDR* até *Dn* (sendo *L1=L2, L1<255* e *L1* tem que ser par);
- Os *bytes AL* e *AH* representam o endereço do início do bloco de memória (*low* e *high*);
- Os *bytes LL* e *LH* indicam o número de *bytes* de dados (no máximo 250, *low* e *high*);
- Os *bytes D1* a *Dn* são os *bytes* de dados;

[28] Imagem retirada de: [Cabrita, 2007]

- Os *bytes CKL* e *CKH* são o *checksum* de 16 *bits* que pode ser calculado através da seguinte fórmula:
  $CHECKSUM = INV[(D_1 + 256.D_2)XOR(D_3 + 256.D_4)XOR \dots XOR(D_{n-1} + 256.D_n)]$
  , ou pelas seguintes:
  $$CKL = INV[(D_1)XOR(D_3)XOR \dots XOR(D_{n-1})]$$
  $$CKH = INV[(D_2)XOR(D_4)XOR \dots XOR(D_n)]$$
- O *byte ACK* é um *byte* de *acknowledge* enviado pelo *BSL* indicando se o comando foi ou não bem sucedido (*DATA_ACK*, que é igual a 0x90, se comando bem recebido e correctamente executado e *DATA_NAK*, que é igual a 0xA0, para erro no *checksum*, comando desconhecido, não permitido ou não executado).

1.4.2.4 - Saída de *BSL*

Para sair de *BSL*, existem duas possibilidades:

- Atavés do comando *Load Program Counter*, é invocado um endereço de memória com o início do programa a ser executado pelo microcontrolador. Neste caso a protecção por palavra-chave não será reactivada;
- Aplicando a sequência de *reset*, representado na *Figura 1.16*, que força o microcontrolador a reiniciar com o *user reset vector* no endereço *0xFFFE*. Para os microcontroladores que têm o pino de *TCK* em vez de pino *TEST*, a sequência é idêntica mas o pino *TCK* deverá estar a *high* em vez de *low*.

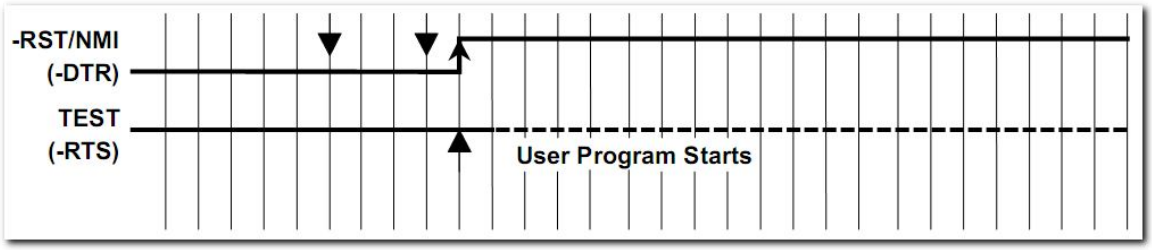


***Figura 1.16 - Sequência de reset*[29]**

1.5 - Estrutura

Esta dissertação está dividida em 8 capítulos.

Neste primeiro capítulo, referente à introdução, foram abordados os temas referentes à contextualização, o estado da arte, os principais componentes utilizados para a elaboração deste projecto, algumas definições importantes para compreender a sua estrutura e os objectivos principais para este projecto. Foi feita uma breve apresentação onde representamos a estrutura

[29] Imagem retirada de: [Texas Instruments, 2001]

do *Pacer2*, os seus principais componentes e descrevemos o *BSL* que é indispensável para a realização de *firmware updates* e que será mencionado ao longo de quase todos os capítulos seguintes.

No segundo capítulo, referente à pista de luzes, iremos expor todo o trabalho já realizado anteriormente, por um dos alunos de projecto, relativo ao *hardware* e *software* desta. Explicaremos o funcionamento global da pista de luzes, os protocolos de comunicação utilizados e descreveremos as ligações entre esta e a buzina. Serão ainda mencionadas todas as correcções e alterações efectuadas tanto a nível de *software* como de *hardware*.

No terceiro capítulo será referido todo o trabalho, realizado pelo mesmo aluno, relativo ao *hardware* e *software* da buzina. O seu funcionamento, os seus protocolos de comunicação e as ligações entre esta e a pista de luzes, bem como as ligações entre o seu microcontrolador e o módulo *wireless* e o *LCD*, serão aqui explicados detalhadamente. Exporemos todos os passos necessários que a buzina tem que efectuar para programar e inicializar correctamente a pista de luzes e serão também mencionadas todas as correcções e alterações efectuadas na buzina a nível de *software* e de *hardware*.

No quarto capítulo será a vez de apresentarmos o *software* e *hardware* herdado do simulador da *PDA*. Explicaremos o seu funcionamento global, cujo principal objectivo é enviar informações e comandos para a buzina a fim de programar e controlar correctamente a pista de luzes, e será fornecida toda a informação necessária para compreender todas as ligações entre os vários componentes deste módulo. Serão ainda mencionadas todas as correcções e alterações efectuadas tanto a nível de *software*, como de *hardware*.

No quinto capítulo apresentaremos todo o *hardware* e *software* herdado do reprogramador da buzina, assim como o seu funcionamento detalhado. O reprogramador é constituído por um simples microcontrolador que será ligado em paralelo com o microcontrolador da buzina ao módulo *wireless*. Este módulo estará por defeito em *LPM (Low Power Mode)* e só activará quando for recebido um comando de início de envio de novo *firmware* para a buzina. Quando este pacote for recebido, o reprogramador activar-se-á, iniciará o modo de *BSL* na buzina e assumirá o controlo do módulo *wireless* a fim de receber os pacotes com o novo *firmware* da buzina e reprogramá-la com este último. No final, o reprogramador da buzina entrará novamente em *LPM* e a buzina ligar-se-á a executar o novo *firmware*. Como consequência, a pista de luzes será também reiniciada e poderá esta também ter sofrido um *firmware update* pois o seu código estará sempre contido no *firmware* da própria buzina. Quer isto dizer que para se fazer um *firmware update* às luzes, é necessário fazer um *firmware update* à buzina para que esta contenha o novo programa das luzes. Ao longo deste capítulo serão ainda mencionadas todas as correcções e alterações efectuadas a nível de *hardware* e de *software*.

O sexto capítulo será o capítulo da apresentação e discussão dos resultados. Aqui serão demonstrados todos os cálculos efectuados na escolha dos diversos componentes a fim de obtermos uma determinada corrente ou queda de tensão necessárias para o bom funcionamento do sistema em geral. Serão apresentados também várias medições e testes efectuados ao funcionamento do mesmo sistema em geral.

No sétimo capítulo serão apresentadas as conclusões finais tendo como análise os objectivos iniciais propostos. Serão ainda discutidos alguns tópicos de trabalho que poderão ser realizados no futuro.

Finalmente no oitavo e último capítulo, será apresentada a bibliografia consultada para a elaboração e escrita desta dissertação.

Capítulo 2. Pista de luzes

2.1 - Introdução

Neste capítulo iremos demonstrar detalhadamente o funcionamento da pista de luzes. Será apresentado todo o trabalho realizado anteriormente e todas as alterações efectuadas durante a realização deste projecto. De realçar que o *software* da pista de luzes disponibilizado no início do projecto não se encontrava a funcionar correctamente.

A pista de luzes é constituída por uma cadeia de *26* luzes. Cada luz é composta por um *LED*, um microcontrolador e uma resistência. As luzes estão ligadas entre si através de um cabo com *1 metro* de comprimento, perfazendo *25 metros* de comprimento total de pista. A *Figura 2.1* mostra as ligações existentes em cada luz.

De modo a tornar a análise mais fácil, iremos simplificar o esquema de cada luz e iremos chamar de *I/O_L* (*input/output left*, pois estão à esquerda da resistência) o pino *P2.1* e de *I/O_R* (*input/output right*, pois estão à direita da resistência) os pinos *P2.0*, *P2.2*, *P2.4* e *P2.5*. Iremos também chamar de *CMP* (*comparator*, pois é o pino da *ADC* que efectua as comparações) o pino *CA0/P2.3*. Na *Figura 2.2* podemos observar as ligações entre todas as luzes que constituem a pista de luzes.

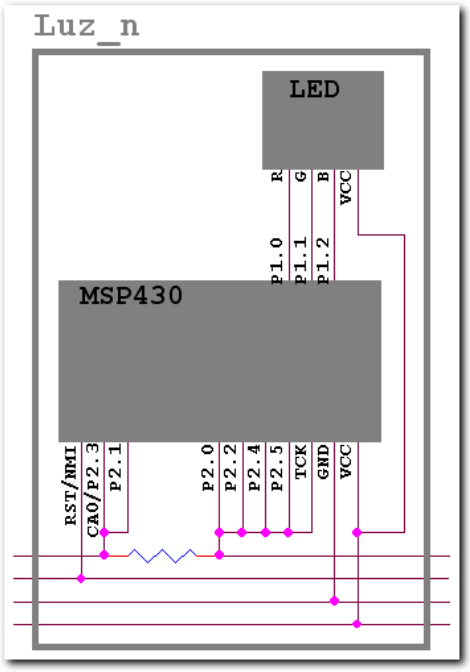


Figura 2.1 – Esquema de cada luz

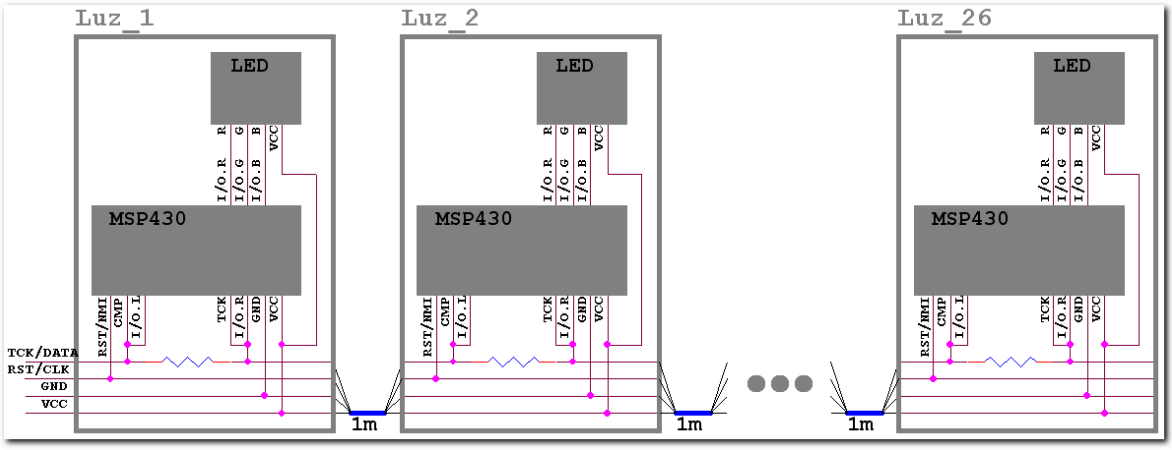


Figura 2.2 - Esquema da Pista de Luzes

Os pinos que constituem o pino *I/O_R* estarão sempre a *input*, excepto aquando da detecção de luzes avariadas. Um desses pinos, o *P2.2*, é o pino utilizado para receber os dados enviados pela buzina por *BSL*. Como as luzes estarão ligadas em paralelo aquando da sua reprogramação por *BSL*, foi decidido ignorar o sinal de *acknowledge* enviada por cada uma pelo pino *P1.1* sinalizando a correcta ou incorrecta recepção dos comandos, caso contrário corria-se o risco de acontecer um curto-circuito ao terminal desses pinos. Assim, partiremos do princípio que todos os comandos enviados por *BSL* serão sempre bem recebidos. Um aspecto relevante é que devido ao facto do pino *P1.1* de cada microcontrolador das luzes estar ligado directamente ao pino que controla a cor verde dos *LED*, estes piscam conforme vão recebendo os *acknowledges* enviados pelo *BSL* de

cada microcontrolador, sendo assim possível observar se todos os microcontroladores estão ou não a ser programados.

A pista de luzes comunicará com a buzina através de um fio de quatro condutores. O condutor *VCC* será portador do nível de referência de alimentação que será fornecido pela buzina; o condutor *GND* será portador do nível de referência de *ground* que também será fornecido pela buzina; o condutor *RST/CLK* ligará o pino *RST/NMI* de cada luz a um pino de *I/O* da buzina e servirá para fazer *reset* ao microcontrolador de cada luz ou transportará o sinal de relógio para a troca de dados entre as luzes e a buzina; o quarto e último condutor será denominado *TCK/DATA* e ligará o pino *TCK* de cada luz a um pino de *I/O* da buzina e servirá para que o microcontrolador entre em modo de *BSL* ou permite a troca de dados entre as luzes e a buzina.

A comunicação entre a Buzina e a Pista de Luzes é síncrona e bi-direccional do tipo *Master-Slave*, em que o *Master* é o microcontrolador da Buzina e os *Slave's* são todos os microcontroladores das Luzes.

Todos os microcontroladores das luzes têm os seus pinos *RST* ligados entre si e os pinos *TCK* também ligados entre si, de modo a activar o *BSL* em simultâneo em todas as luzes. O pino *RST* é também o pino das *NMI* (*Non-Maskable Interrupt*) e é por isso usado para receber o sinal de *clock* enviado pela buzina, aquando do envio de comandos para as luzes ou do envio de informações para a buzina. Esta rotina de atendimento à *NMI* será explicada detalhadamente mais à frente neste capítulo.

O pino *CMP*[30] é o pino de entrada do comparador que será utilizado na detecção de luzes avariadas e estará sempre a *input*. Este pino será utilizado apenas na detecção de luzes avariadas. Este processo será explicado mais adiante neste capítulo. O pino de *I/O_L* ligado ao *CMP* é o pino *P2.1* que será utilizado para ler o bit de dados.

O *LED* será controlado com 4 fios, um para o *VCC* e os outros 3 seleccionam a(s) cor(es) activa(s). O *VCC* do *LED* liga directamente ao *VCC* fornecido pela buzina, enquanto os 3 pinos que seleccionam cada cor estão ligados a 3 pinos de *I/O* do microcontrolador a ele associado[31]. Todos os pinos do *LED* que seleccionam as diferentes cores funcionam em lógica negativa[32].

O sinal de *clock* é gerado pelo *Master* sempre que seja necessário transferir dados e é partilhado por todos os *Slave's*. Quando o *Master* envia informação, esta chega a todos os *Slave's* e é da responsabilidade destes filtrar essa informação e concluir se esta é destinada a eles ou não. Já os *Slave's* só enviam dados quando tal assim for solicitado pelo *Master* e só um *Slave* pode transmitir informação de cada vez. Obviamente, enquanto um *Slave* transmite informação, a buzina também não pode transmitir informação.

---

[30] Também denominado pino CA0/*P2.3*.
[31] O pino *I/O.R* é o pino *P1.0* e controla o vermelho, o pino *I/O.G* é o pino *P1.1* e controla o verde e o pino *I/O.B* é o pino *P1.2* e controla o azul.
[32] Estão activos quando a sua entrada é *low* e inactivos quando a sua entrada é *high*.

A comunicação é efectuada, como explicado anteriormente, através de 2 fios condutores, um para o sinal de *clock* e o outro para o envio em série dos dados. Como o sinal de *clock* é sempre gerado pelo *Master*, esta linha é unidireccional, mas como os dados podem ser transmitidos tanto pelo *Master*, como pelos *Slave's*, esta linha é bidireccional.

A rotina de leitura do *MSP430* da luz demora 94μs a ler um *bit*, sendo que no último *bit* demora 120μs, pelo que impomos um sinal de *clock*, no mínimo, com um período de 200μs, ou seja, uma frequência máxima de 5.0*KHz*, tal como se pode ver pela *Figura 2.3* e *Figura 2.4*.

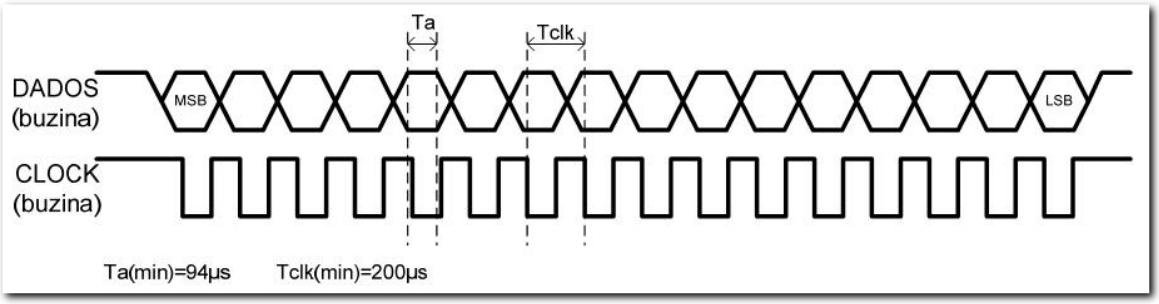


Figura 2.3 - Transferência de dados entre a Buzina e as Luzes[33]

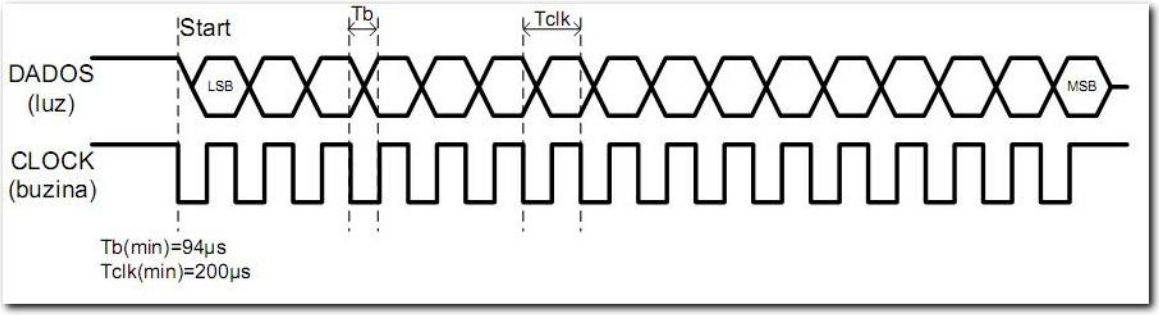


Figura 2.4 - Transferência de dados entre as Luzes e a Buzina[34]

2.2 - Rotina de atendimento à NMI

A troca de informação entre as luzes e a buzina é efectuada, nas luzes, através de interrupções. O sinal de *clock* fornecido pela buzina provoca uma *NMI* e é a sua rotina de atendimento que processa a informação recebida/enviada.

A primeira coisa que essa rotina faz é verificar se se está a enviar ou receber informação. Se estiver a receber informação, coloca os *bits* recebidos um a um num *buffer* para isso destinado, se estiver a enviar informação então essa informação já tem que estar previamente colocada no

[33] Imagem retirada de: [Cabrita, 2007]
[34] Imagem retirada de: [Cabrita, 2007]

buffer e o que a rotina faz é enviar os *bits* um a um para a buzina. Toda a vez que a rotina de atendimento à *NMI* é invocada, apenas um *bit* é enviado/recebido.

Na *Figura 2.5* podemos ver o diagrama da rotina de atendimento à *NMI*.

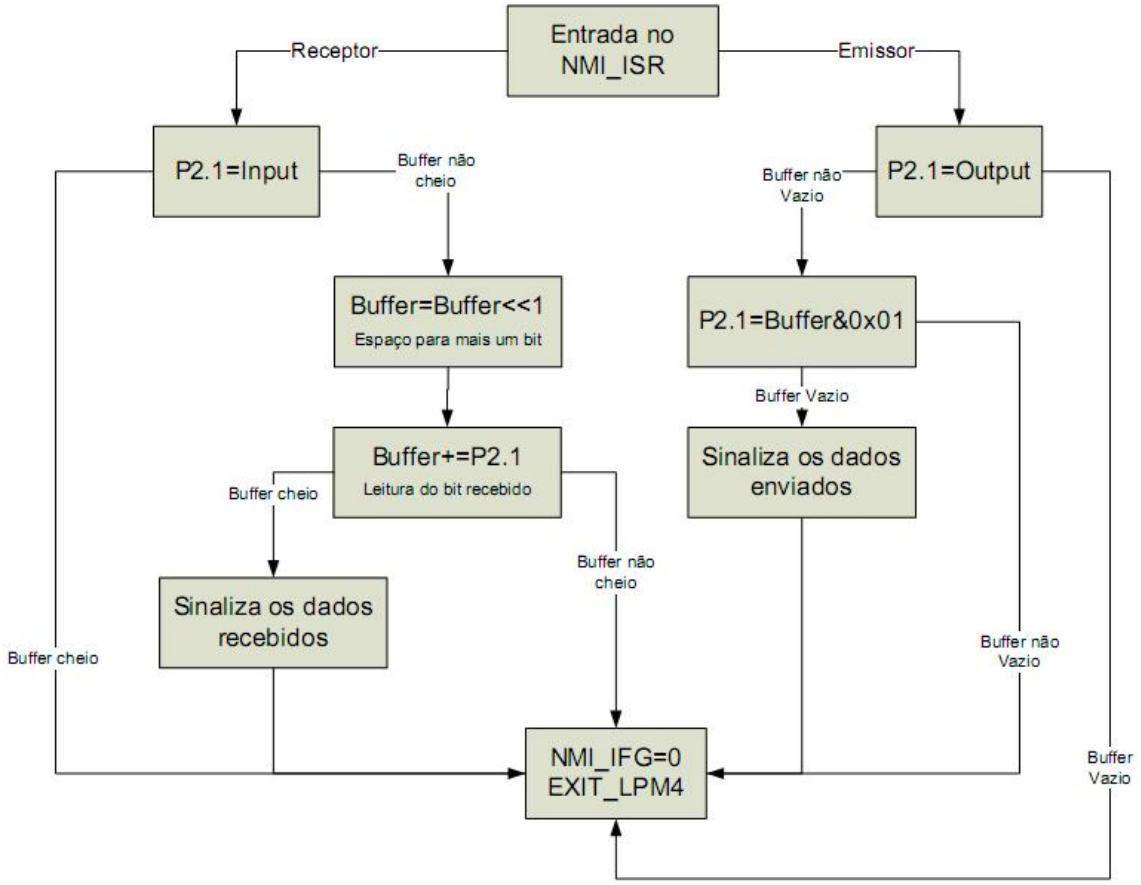


Figura 2.5 - Diagrama da rotina de atendimento à NMI[35]

2.3 - Comandos das luzes

Para controlar as luzes, foram definidos oito comandos diferentes. Estes comandos são suficientes para executar todas as acções necessárias ao bom funcionamento da pista. Ao pacote de dados que contém um comando chamaremos de trama.

Os dados são enviados em blocos de 16*bits* (2*bytes*). Existem 3 campos na trama, são eles: o endereço, que identifica o *ID* do(s) destinatário(s) da informação; o comando, que identifica o comando a ser executado; os dados, onde estarão contidos os dados relevantes para a execução do comando especificado.

[35] Imagem retirada de: [Cabrita, 2007]

A posição dos vários campos na trama está relacionada com o número de operações necessárias para tratar cada um deles, uma vez que serão necessárias máscaras e operações de deslocamento de *bit*. Dito isto, e como o único campo que terá que ser sempre tratado por todas as luzes é o campo que contém o endereço (todas as luzes têm que ler este campo, de modo a descobrir se os dados são destinados a si ou não), este está posicionado nos *bits* menos significativos da trama, deste modo, o endereço não necessita de ser deslocado nenhum *bit*. Como o campo de dados não é utilizado em alguns comandos, e como faz todo o sentido que o comando seja o próximo campo a ser lido, o comando será colocado nos *bits* intermédios da trama, fazendo com que sejam necessários menos deslocamentos para determiná-lo do que para retirar os dados. Estes serão assim colocados nos *bits* mais significativos da trama, exigindo mais operações de deslocamento do que qualquer outro campo.

O campo de endereço será composto por 5*bits* (*bit0* a *bit4*), pois é o mínimo necessário para referenciar as 26 luzes que constituem a Pista de Luzes. Podemos ainda reservar desta forma o endereço 0*x*1F para *broadcast* e o endereço 0*x*00 para a Buzina. O endereço de *broadcast* permitirá que um comando seja executado em simultâneo por todas as luzes e o endereço reservado para a buzina garante que nenhuma luz aceite aqueles dados por engano, o que teria consequências imprevisíveis no funcionamento da mesma[36].

O campo de dados será composto por 7*bits* (*bit9* a *bit15*), pois o maior número que será necessário transmitir será o 127_{10}, mais à frente será explicado o porquê deste valor.

O campo do comando será composto pelos restantes 4*bits* (*bit5* a *bit8*), o que possibilita um máximo de 16 comandos distintos, que serão suficientes para as tarefas que as luzes terão que executar, tal como poderá ser comprovado mais à frente quando definirmos todos eles.

A trama fica assim definida como mostra a *Figura 2.6*.

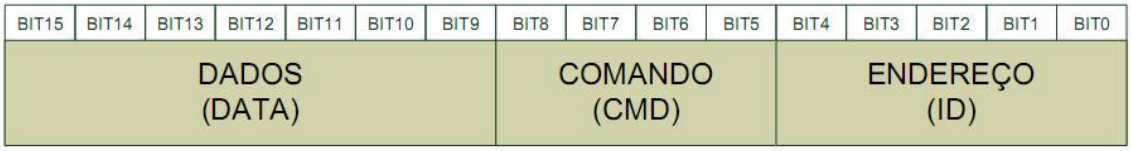


Figura 2.6 - Campos da trama[37]

A *Tabela 2.1* apresenta os oito comandos disponíveis para controlar a pista de luzes. Iremos de seguida explicar detalhadamente o funcionamento de cada um deles.

[36] As consequências dependeriam dos dados que estariam a ser transmitidos naquele instante para a Buzina
[37] Imagem retirada de: [Cabrita, 2007]

Comando (CMD)	Definição
0x01	Selecção de cor
0x02	Dimensionamento de PWM
0x03	Limitação de luminescência
0x04	Acender LED
0x05	Apagar LED
0x06	Configuração da detecção de avaria
0x07	Activar detecção de avaria
0x08	Atribuição manual de ID

Tabela 2.1 - Comandos disponíveis

2.3.1 - Selecção de cor

O comando selecção de cor, selecciona a(s) cor(es) que o *LED* apresentará quando ligado.

Deverá conter no campo de dados a(s) cor(es) que se pretende(m) activar e no campo do endereço o *ID* da(s) luz(es) que se quer(em) configurar.

A cor será definida nas luzes por uma variável do tipo *char*[38], em que os 3*bits* menos significativos correspondem cada um a uma cor diferente (*bit0* para o vermelho, *bit*1 para o verde e *bit*2 para o azul).

Este comando escreve na variável que representa a cor, o valor enviado no campo de dados. Para seleccionar a cor, o seu *bit* deverá estar a 1, em caso contrário, esta cor não será seleccionada.

Este comando fica definido com o valor 0x01 e a estrutura da trama pode ser observado na *Figura 2.7*.

Tempo de execução: 15μs.

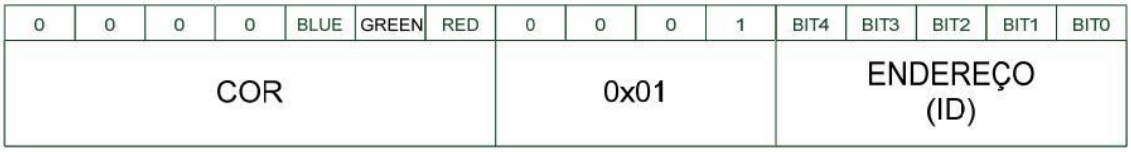


Figura 2.7 - Comando 0x01 - Selecção de cor[39]

Exemplo de utilização:

Seleccionar a cor verde e azul na luz com *ID* igual a 8 ---------- 0000110 0001 00100

[38] Tipo *char* ocupa 1*byte*
[39] Imagem retirada de: [Cabrita, 2007]

Seleccionar a cor vermelha em todas as luzes ---------- 0000001 0001 11111

2.3.2 - Dimensionamento de PWM

O comando dimensionamento do *PWM*, atribui o valor, em percentagem, de *duty cycle* do *PWM* da(s) cor(es) previamente seleccionada(s). Com a implementação do *duty cycle* conseguimos controlar facilmente a intensidade luminosa de cada *LED*.

Deverá conter no campo de dados o valor correspondente à percentagem do *duty cycle* (0% - 100%) e no campo do endereço o *ID* da(s) luz(es) que se quer(em) configurar.

Este comando fica definido com o valor 0x02 e a estrutura da trama pode ser observado na *Figura 2.8.*

Tempo de execução: 120μs.

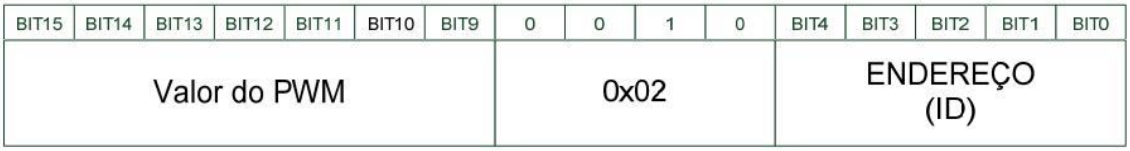


Figura 2.8 - Dimensionamento do PWM[40]

Exemplo de utilização:

Define o *duty cycle* da luz com *ID* igual a 14 em 80% ---------- 1010000 0010 01110
Define o *duty cycle* da luz com *ID* igual a 26 em 10% ---------- 0001010 0010 11010

2.3.3 - Dimensionamento de luminescência

Num *LED RGB*, e para um determinado valor de tensão de alimentação, as três cores tendem em apresentar luminosidades diferentes. É por isso necessário existir um comando que uniformize a intensidade luminosa das(s) cor(es) previamente seleccionada(s). Este comando irá interferir directamente com o *duty cycle* criado aquando do dimensionamento de *PWM* de forma a uniformizar a luminescência das várias cores.

Deverá conter no campo de dados um valor compreendido entre *1* e *127* correspondentes ao mínimo e máximo de luminescência permitida para uma determinada cor. É este valor, *127*, que limita o tamanho do campo de dados. No campo do endereço deverá estar contido o *ID* da(s) luz(es) que se quer(em) configurar. Este comando deve ser enviado em *broadcast* sempre que as luzes apresentem a mesma cor seleccionada, caso contrário, seleccionam-se, um a um, os *ID* das luzes que emitam essa cor.

[40] Imagem retirada de: [Cabrita, 2007]

Este comando fica definido com o valor 0x03 e a estrutura da trama pode ser observado na *Figura 2.9*.

Tempo de execução: 120μs.

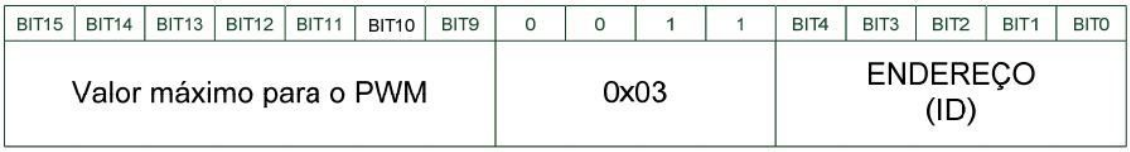


Figura 2.9 - Valor máximo de potência da cor[41]

Exemplo de utilização:

Define o valor máximo de *PWM* de todas as luzes em 64/127 --------- 1000000 0011 11111

2.3.4 - Acender LED

O comando acender *LED*, acende o(s) *LED* seleccionado(s) com a(s) respectiva(s) cor(es), *PWM* e consequente(s) intensidade(s).

Este comando não necessita de dados e deverá conter no campo endereço o *ID* da(s) luz(es) que se quer(em) configurar.

Este comando fica definido com o valor 0x04 e a estrutura da trama pode ser observado na *Figura 2.10*.

Tempo de execução: 700μs.

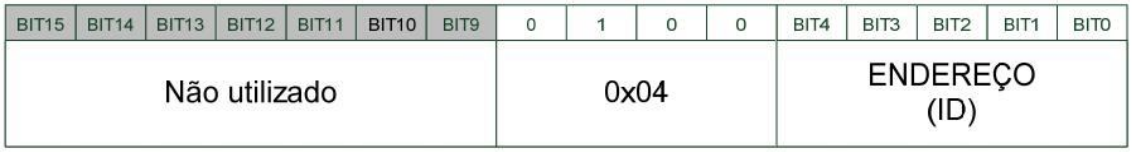


Figura 2.10 - Acender LED[42]

Exemplo de utilização:

Acende o *LED* da luz com *ID* igual a 11 ---------- 0000000 0100 01011
Acende o *LED* da luz com *ID* igual a 19 ---------- 0000000 0100 10011

2.3.5 - Apagar LED

O comando apagar *LED*, apaga o(s) *LED* seleccionado(s), mantendo inalterado a variável que armazena a cor e os valores anteriormente definidos de *PWM* e de potência da cor.

[41] Imagem retirada de: [Cabrita, 2007]
[42] Imagem retirada de: [Cabrita, 2007]

Este comando não necessita de dados e deverá conter no campo endereço o *ID* da(s) luz(es) que se quer(em) configurar.

Este comando fica definido com o valor 0*x*05 e a estrutura da trama pode ser observado na *Figura 2.11*.

Tempo de execução: 130μs.

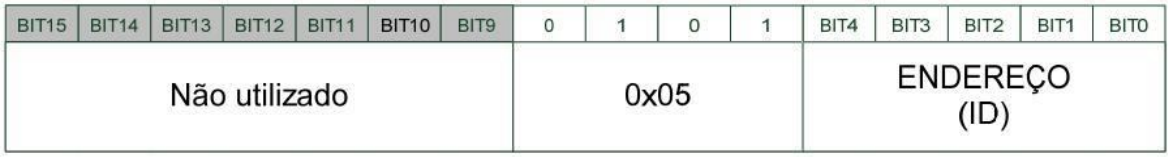


Figura 2.11 - Apagar LED[43]

Exemplo de utilização:

Apaga o *LED* da luz com *ID* igual a 3 ---------- 0000000 0101 00011
Apaga o *LED* de todas as luzes ---------- 0000000 0101 11111

2.3.6 - Configuração de detecção de avaria

Se, por alguma razão, o microcontrolador de uma luz deixar de funcionar, é necessário detectá-lo de modo a que este não interfira no correcto endereçamento das restantes.

O comando configuração de detecção de avaria configura o papel que determinada luz irá tomar quando se activar a detecção de microcontroladores avariados. Existem três estados que podem ser configurados em cada luz individualmente: estado activo, estado passivo e estado inactivo. Apenas uma luz pode estar configurada com o estado activo de cada vez e a luz imediatamente anterior a esta deve obrigatoriamente estar configurada com o estado passivo, caso contrário a detecção de avaria não funcionará correctamente quando for activada.

No final da execução do comando de detecção de avaria, as luzes que foram utilizadas para essa detecção devem ser sempre configuradas novamente para o estado inactivo de modo a não interferirem na posterior detecção de avarias entre outras luzes.

Deverá conter no campo de dados o valor correspondente ao papel que determinada luz terá na detecção de avaria: se tiver 0, a luz será configurada com o estado inactivo; se tiver 1, a luz será configurada com o estado activo; se tiver 2, a luz estará configurada com o estado passivo. No campo endereço deverá conter o *ID* da(s) luz(es) a configurar.

Este comando fica definido com o valor 0*x*06 e a estrutura da trama pode ser observado na *Figura 2.12*.

Tempo de execução: 100μs.

[43] Imagem retirada de: [Cabrita, 2007]

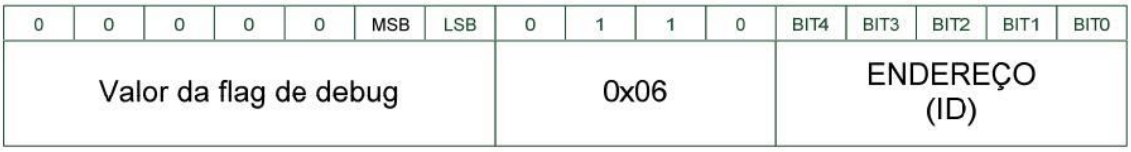


Figura 2.12 - Configuração do estado de detecção de avaria[44]

Exemplo de utilização:

Configura a luz com *ID* igual a 7 com papel activo ---------- 0000001 0110 00111
Configura todas as luzes com papel inactivo ---------- 0000000 0110 11111

2.3.7 - Activar detecção de avaria

O comando activar a detecção de avaria, desencadeia o processo de detecção de microcontroladores avariados. A luz configurada com o papel passivo colocará os pinos à direita da sua resistência a *output low* e espera o tempo suficiente para que a luz configurada com estado activo faça a leitura do comparador. A luz configurada então com o papel activo colocará os pinos à direita da sua resistência a *output high* e efectua a leitura do seu comparador, enviando de seguida para a buzina o valor lido (será assim necessário efectuar uma leitura por parte da buzina sempre que este comando seja efectuado).

Este comando não necessita de dados e deverá conter no campo endereço o *ID* de *broadcast*.

Este comando só deverá ser enviado em *broadcast*, caso contrário a detecção de avaria não funcionará.

Este comando fica definido com o valor 0*x*07 e a estrutura da trama pode ser observado na *Figura 2.13*.

Tempo de execução em estado inactivo: 2.56*ms*.

Tempo de execução em estado passivo: 3.6*ms*.

Tempo de execução em estado activo: 3.8*ms*.

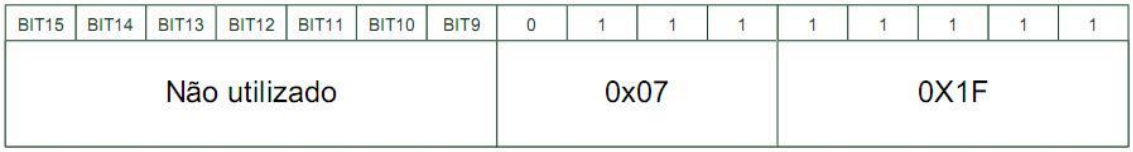


Figura 2.13 - Activar a detecção de avaria[45]

Exemplo de utilização:

Activa a detecção de avaria ---------- 0000000 0111 11111

[44] Imagem retirada de: [Cabrita, 2007]
[45] Imagem retirada de: [Cabrita, 2007]

2.3.8 - Atribuição manual de ID

O comando atribuição manual de *ID* permite alterar o *ID* de determinada luz. Este comando deverá ser utilizado sempre que se detectarem luzes avariadas de modo a corrigir o *ID* das luzes posteriores permitindo um normal funcionamento da pista. O *ID* das luzes deve ser sempre corrigido por ordem decrescente, caso contrário todas as luzes alteradas ficarão com o mesmo *ID* (que será o *ID* da última luz).

Deverá conter no campo de dados o novo *ID* a atribuir e no campo do endereço o *ID* da luz que se quer alterar.

Por razões óbvias, este comando nunca deverá ser utilizado em *broadcast*.

Este comando fica definido com o valor 0*x*08 e a estrutura da trama pode ser observado na *Figura 2.14*.

Tempo de execução: 15μs.

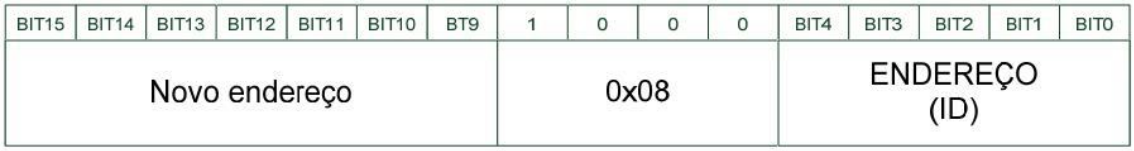


***Figura 2.14 - Atribuição manual de ID*[46]**

Exemplo de utilização:

Altera o *ID* da luz 7 para 8 ---------- 0001000 1000 00111
Altera o *ID* da luz 25 para 26 ---------- 0011010 1000 11001

O anexo A contém todos os comandos disponíveis para programar correctamente as luzes. Aconselhamos a sua consulta sempre que seja necessário enviar algum comando para as luzes.

2.4 - Firmware updates através de BSL

Na *Figura 2.15* temos um esquema simplificado da ligação entre a buzina e os microcontroladores de cada luz. Para iniciar o *BSL*, a buzina coloca a sequência de inicialização nos pinos *RST* e *TCK*, imediatamente todos os microcontroladores das luzes ficam com os pinos a input e dessa forma o sinal flui através das resistências até ao último microcontrolador, programando-os simultaneamente.

[46] Imagem retirada de: [Cabrita, 2007]

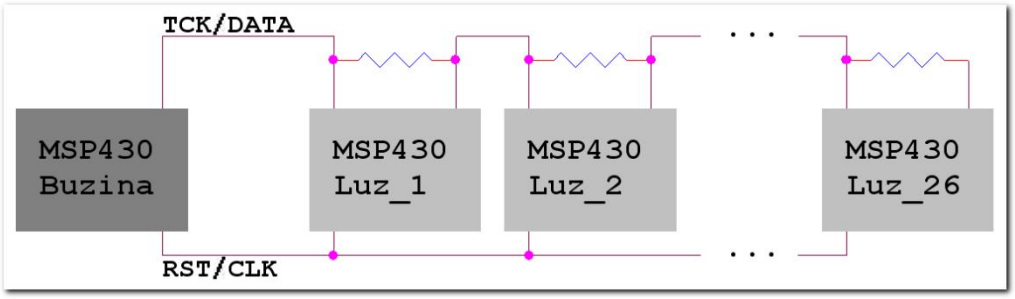


Figura 2.15 - Esquema de bootstrap

No final do *BSL*, os microcontroladores iniciam normalmente e colocam todos os pinos *I/O_L* a *input* e os pinos *I/O_R* a *output low*, de modo a impedir que o sinal se propague para que seja efectuado de seguida o endereçamento.

Será através do *BSL* que será possível actualizar o firmware das luzes. Sempre que o sistema é ligado, a buzina conterá no seu microcontrolador o código hexadecimal do programa das luzes e este será enviado por *BSL* em paralelo para todas as luzes pelo método explicado com a ajuda da *Figura 2.15*.

2.5 - Endereçamento das luzes

O endereçamento das luzes é a primeira coisa que estas fazem quando ligam, de modo a que cada uma fique com um *ID* próprio e diferente de todas as outras para se poderem individualizar e receber comandos que apenas a elas são destinadas.

O modo de endereçamento pode ser analisado com a ajuda da *Figura 2.16*. A comunicação é efectuada através da linha *TCK/DATA*. Inicialmente, os microcontroladores estão configurados de modo a impedir que o sinal se propague de uns para os outros. Isso é conseguido, colocando os pinos *I/O_R* a *output low* e os pinos *I/O_L* a *input*.

Cada microcontrolador terá uma variável interna denominada *myID* que será utilizada para guardar o valor do seu *ID* e que é inicializada em todos com o valor de 1.

A buzina envia um impulso largo para a primeira luz a sinalizar o início do endereçamento. A *Figura 2.17* mostra a buzina a enviar o impulso largo para a primeira luz, as setas indicam o sentido da comunicação e um *I* indica que o pino está configurado como *input* e um *O* como *output*.

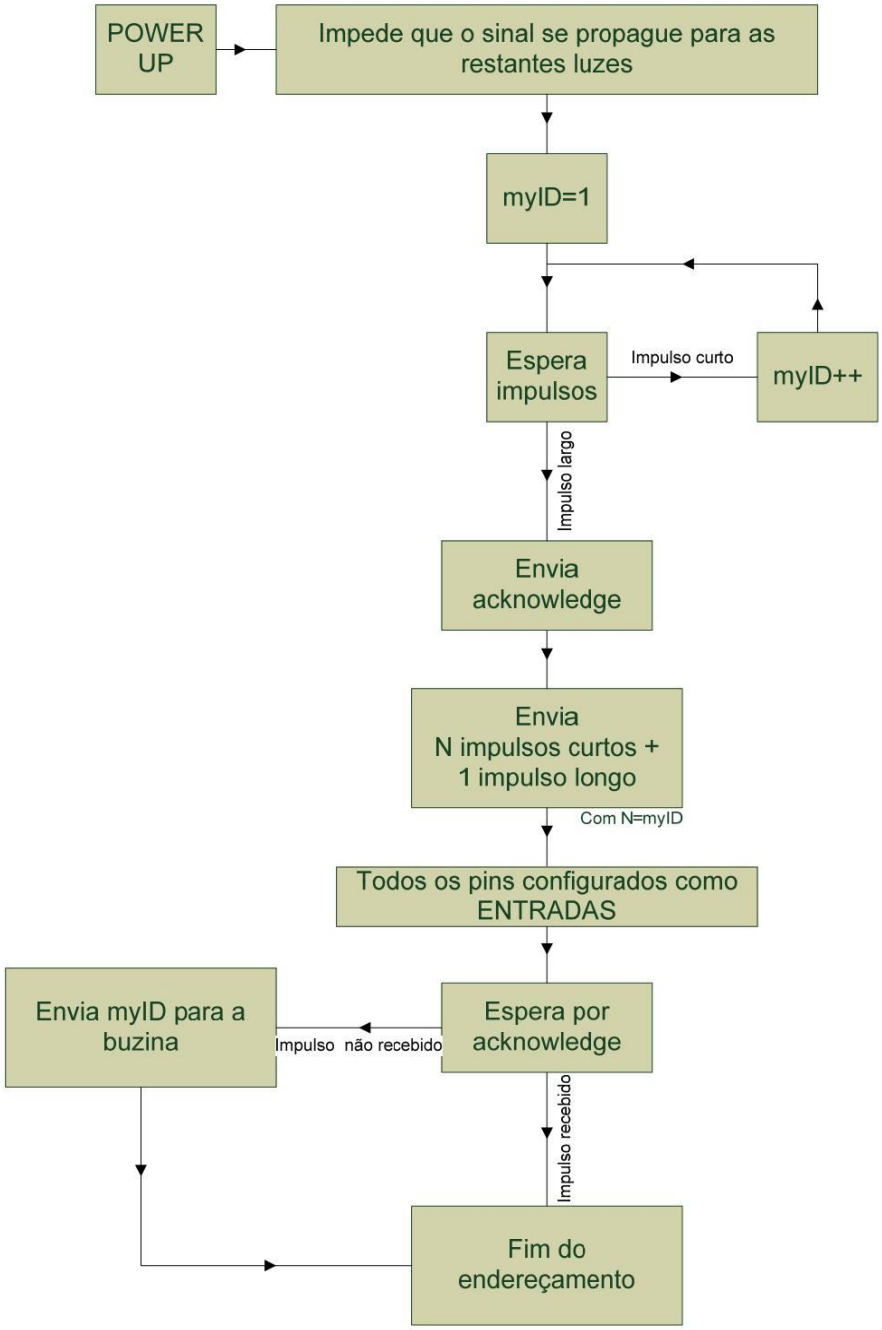


Figura 2.16 - Algoritmo de endereçamento das luzes[47]

[47] Imagem retirada de: [Cabrita, 2007]

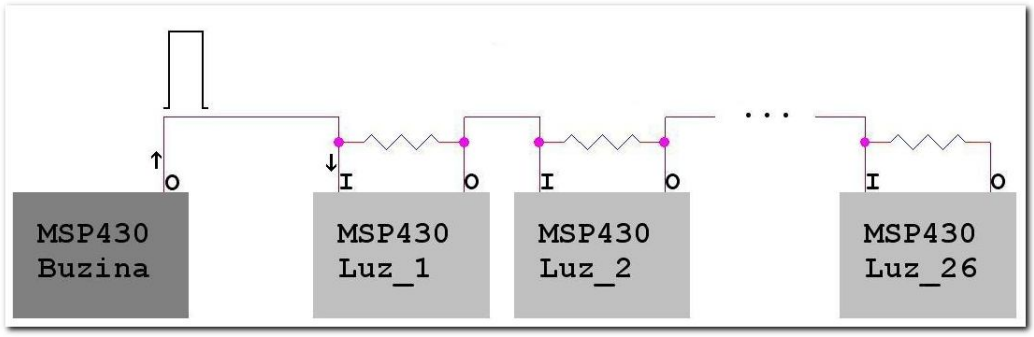


Figura 2.17 – Inicialização do endereçamento por parte da buzina

O microcontrolador da primeira luz recebe o impulso no seu pino *I/O_L* e assume então que o seu *ID* é igual a 1 e envia, através dos pinos *I/O_R*, um impulso curto que representa o seu *ID* e um impulso longo que sinaliza o final do envio do seu valor de *ID*, traduzido em impulsos curtos. A *Figura 2.18* mostra o envio dos impulsos para a segunda luz.

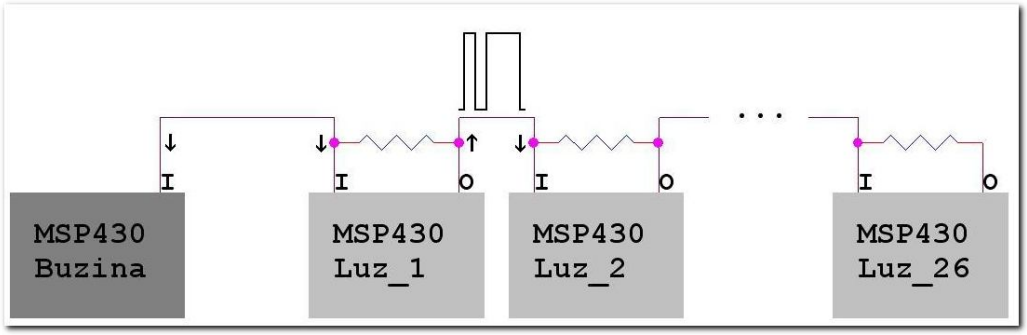


Figura 2.18 – Endereçamento da primeira luz

O microcontrolador da segunda luz, recebe então um impulso curto e incrementa o seu *ID* para 2. Depois recebe um impulso largo e assume que o seu endereçamento está concluído e envia um impulso de *acknowledge* para o microcontrolador da luz anterior a sinalizar que recebeu os impulsos correctamente como demonstrado na *Figura 2.19*.

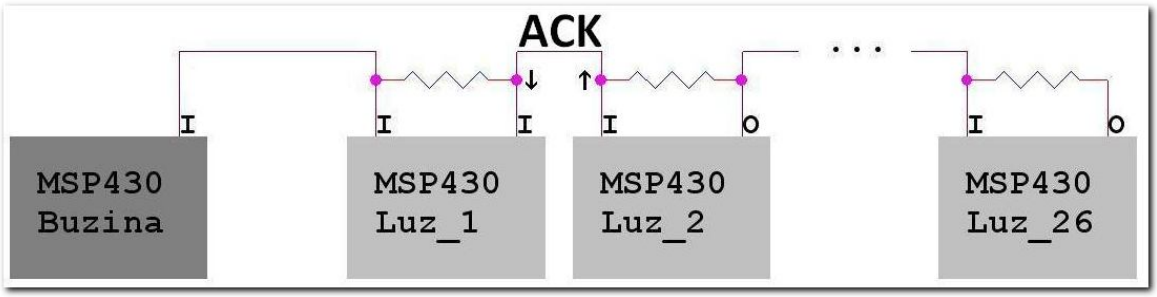


Figura 2.19 - Envio do sinal de acknowledge para a primeira luz

De seguida, envia para o terceiro microcontrolador dois impulsos curtos e um impulso longo e aguarda pelo respectivo *acknowledge como* demonstrado na *Figura 2.20*. O terceiro microcontrolador recebe dois impulsos curtos e incrementa o seu *ID* para 3. Recebe um impulso longo e assume que o seu endereçamento terminou e envia o sinal de *acknowledge* para a segunda luz.

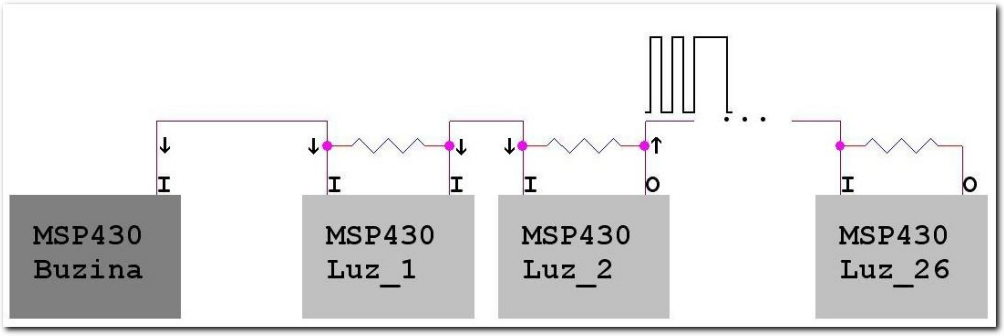


Figura 2.20 - Endereçamento da segunda luz

Este ciclo repete-se até que a última luz ficará um determinado tempo de *timeout* à espera de um *acknowledge* por parte de um eventual microcontrolador da luz seguinte que não existe. Esgotando-se esse tempo de *timeout*, a luz assume que é a última luz da pista e envia o valor do seu *ID* de volta para a buzina como mostra a *Figura 2.21*.

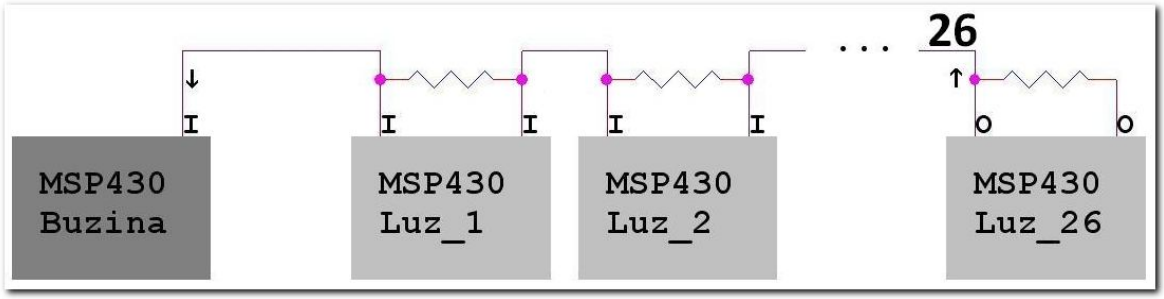


Figura 2.21 - Envio do ID da última luz para a buzina

2.5.1 - Alterações efectuadas

Analisado o algoritmo original do modo de endereçamento das luzes, foi necessário efectuar algumas alterações com a finalidade de o tornar mais rápido e simples, reduzindo ainda o tamanho correspondente deste bloco de código. Como foi apenas alterado o algoritmo, não houve necessidade de alterar qualquer ligação, sendo que o endereçamento continua a ser efectuado através da mesma linha de dados que anteriormente.

O novo algoritmo é em tudo semelhante ao inicial, mas foi abandonada a ideia do envio dos *acknowledges* por parte de cada luz. Tal procedimento não é necessário, pois durante todo o processo de endereçamento a buzina estará à 'escuta' e conseguirá determinar com exactidão o

número de luzes presentes na pista, bastando para isso contar o número de impulsos longos. Como cada luz envia apenas um impulso longo (podendo no entanto enviar vários impulsos curtos, dependendo do seu *ID*) o número de impulsos longos detectados pela buzina correspondem ao número total de luzes da pista.

Na *Figura 2.22*, podemos visualizar o novo algoritmo utilizado no endereçamento das luzes de uma forma mais simples e directa.

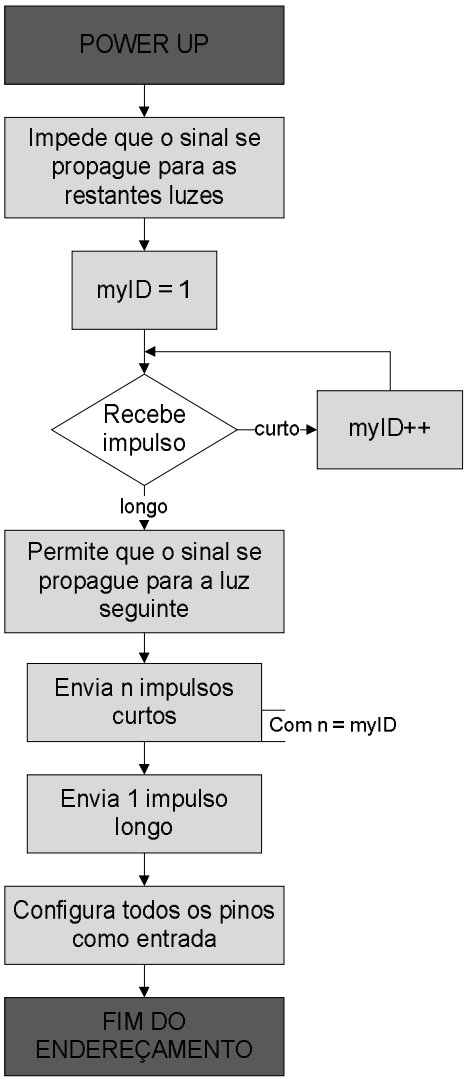


Figura 2.22 - Modo de endereçamento das luzes

Inicialmente todos os microcontroladores colocam os pinos *I/O_L* a *input* e os pinos *I/O_R* são colocados a *output low*, desta forma não há passagem de sinal de um microcontrolador para o outro. Cada microcontrolador tem também uma variável designada por *myID* que é inicializada a 1.

Para sinalizar o inicio de endereçamento das luzes, a buzina envia um impulso longo que será recebido apenas pelo microcontrolador da primeira luz como mostrado na *Figura 2.23*.

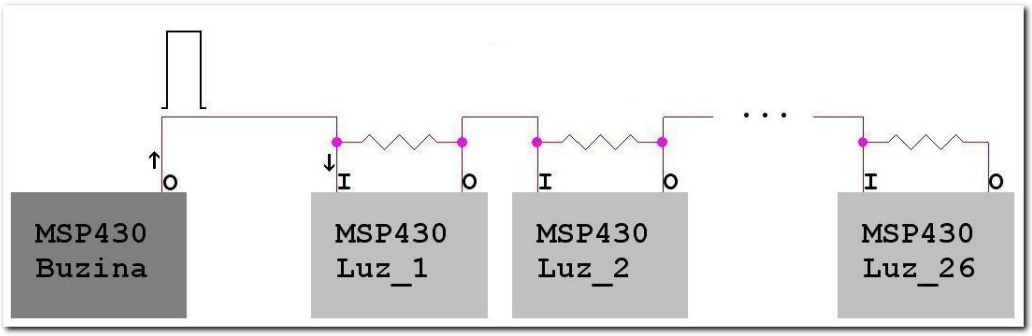


Figura 2.23 - Inicialização do endereçamento por parte da buzina

A buzina coloca de seguida o seu pino a *input* e estará a partir deste momento à 'escuta' por impulsos longos. O microcontrolador da primeira luz ao receber um impulso longo no seu pino *I/O_L*, assume que o endereçamento começou e envia através do pino *I/O_R* um impulso curto, que representa o seu *ID* (neste caso *myID* é igual a 1), e finaliza enviando um impulso longo. De seguida coloca os seus pinos *I/O_R* também eles a *input*. De notar que estes impulsos apenas serão recebidos pelo microcontrolador seguinte e pela própria buzina como demonstrado na *Figura 2.24*.

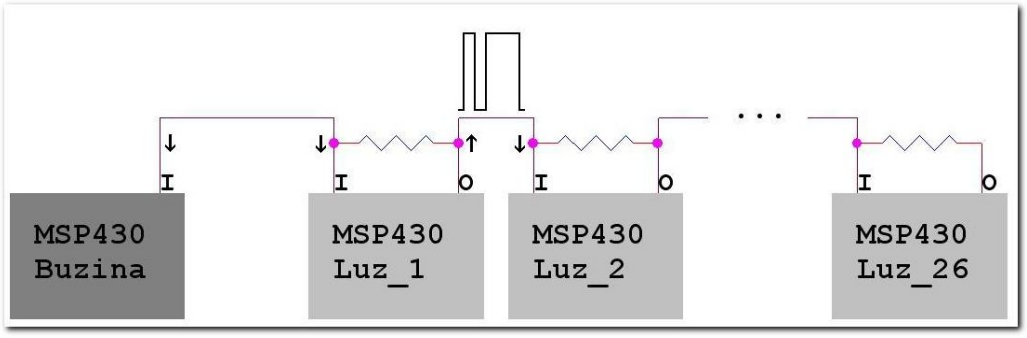


Figura 2.24 - Endereçamento da primeira luz

O microcontrolador da segunda luz recebe então um impulso curto e um longo. Por cada impulso curto que um microcontrolador receba, ele incrementa o seu *ID* e espera até receber um impulso longo a sinalizar o final de envio do valor do *ID* do microcontrolador anterior (neste caso *myID* ficará igual a 2, pois recebeu um impulso curto seguido imediatamente de um longo).

De seguida, envia dois impulsos curtos que representam o seu *ID* e finaliza enviando um impulso longo. Estes impulsos serão recebidos pelo microcontrolador da luz seguinte, por todos os microcontroladores anteriores e pela buzina. Os microcontroladores anteriores, como já têm *ID* atribuído, ignorarão os impulsos. No final, coloca os seus pinos *I/O_R* a *input*. Na *Figura 2.25* podemos observar o momento em que a segunda luz envia os impulsos que serão recebidos pela buzina, pelo primeiro microcontrolador e pelo terceiro microcontrolador.

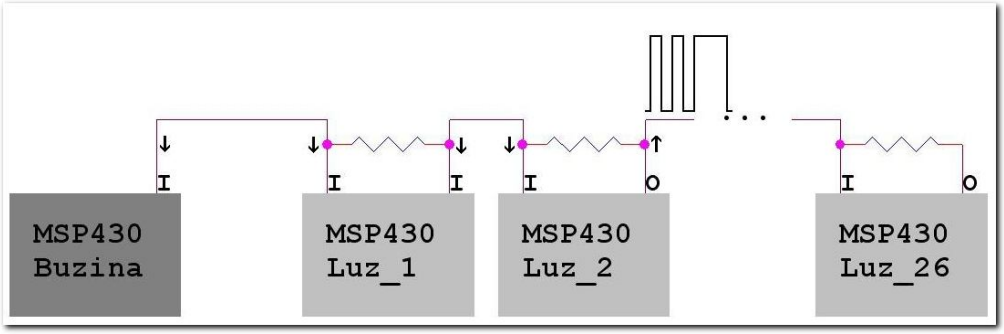


Figura 2.25 - Endereçamento da segunda luz

O microcontrolador da terceira luz recebe então dois impulsos curtos, incrementando duas vezes o seu *ID* (ficará portanto com *myID* igual a 3), seguido de um impulso longo. E este ciclo é repetido novamente até chegar à última luz.

A buzina sabe o número total de luzes que constituem a pista, ao contar o número de impulsos longos que recebe durante o endereçamento das luzes. Como cada microcontrolador só envia um impulso longo (independentemente do número de impulsos curtos que envie) esta é uma forma simples e suficiente para determinar correctamente o número total de luzes da nossa pista. A buzina sabe que o endereçamento acabou, depois de esgotar um determinado tempo de *timeout* sem receber qualquer tipo de impulso, curto ou longo.

2.6 - Detecção de luzes avariadas

Segundo o *datasheet* do microcontrolador (família *MSP430x1xx*), este possui as duas entradas do seu comparador acessíveis através dos pinos *P2.3* (*CA0*) e *P2.4* (*CA1*). Estas entradas podem também ser ligadas internamente a alguns valores de referência, tais como $VCC/2$ ou $VCC/4$.

Para prevenir a eventualidade de mau funcionamento de um dado microcontrolador, o que determinaria o incorrecto endereçamento e utilização das luzes subsequentes, houve a necessidade de desenvolver um método para detectar esses microcontroladores avariados de uma forma completamente automática.

Essa detecção será efectuada sempre que se inicializa a pista de luzes e será realizada logo depois do endereçamento das mesmas.

Para detectar um microcontrolador avariado, iremos utilizar o comparador existente no segundo de dois microcontroladores com *ID* consecutivos, configurado para comparar o valor lido no pino *CA0* com $VCC/4$. O comparador devolve o valor 0 se o valor lido for inferior à tensão de referência (neste caso, $VCC/4$) ou devolve 1 se o valor lido for superior.

Sabendo isto, colocaremos uma tensão igual a *GND* nos *I/O_R* do enésimo microcontrolador e uma tensão igual a *VCC* nos pinos *I/O_R* do microcontrolador seguinte ao enésimo. De seguida efectuamos a leitura do comparador que está conectado ao pino *I/O_L* do microcontrolador seguinte ao enésimo.

Se não existir nenhum microcontrolador avariado entre dois microcontroladores com *ID* consecutivos, haverá um curto-circuito entre os pinos *I/O_R* do enésimo microcontrolador e o pino *I/O_L* do microcontrolador seguinte, fazendo com que a tensão colocada no comparador seja a tensão colocada nos pinos *I/O_R* do enésimo microcontrolador, ou seja, *GND*. Como $GND < VCC/4$, o comparador devolverá o valor 0. Na *Figura 2.26* podemos ver um exemplo da detecção de avaria entre duas luzes com *ID* consecutivos (neste caso a primeira tem *ID* igual a 1 e o segunda tem *ID* igual a 2), sem nenhum microcontrolador avariado entre elas. Na *Figura 2.27* podemos ver a queda de tensão na resistência em pormenor e o valor que será lido pelo comparador.

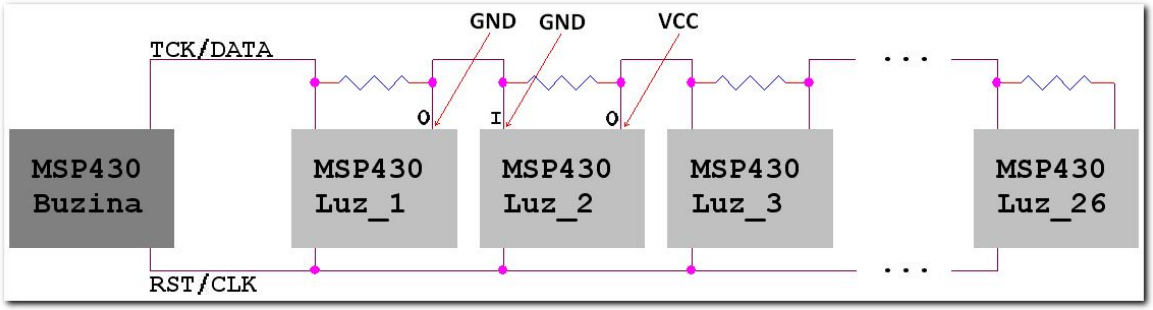


Figura 2.26 - Detecção de avaria em duas luzes consecutivas, sem nenhuma avariada entre elas

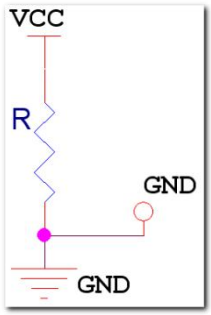


Figura 2.27 - Queda de tensão na resistência e valor lido no comparador

Se existir um microcontrolador avariado entre dois microcontroladores com *ID* consecutivos, haverá uma resistência entre os pinos *I/O_R* do enésimo microcontrolador e o pino *I/O_L* do microcontrolador com *ID* seguinte, fazendo com que haja uma queda de tensão de $VCC/2$ na resistência do microcontrolador avariado e colocando assim a tensão de $VCC/2$ no pino *I/O_L* do microcontrolador com *ID* seguinte ao enésimo microcontrolador. Como $VCC/2 > VCC/4$, o comparador devolverá o valor 1. Na *Figura 2.28* podemos ver um exemplo da detecção de avaria entre duas luzes com *ID* consecutivos (neste caso a primeira tem *ID* igual a 1 e a segunda tem *ID*

igual a 2), com um microcontrolador avariado entre elas. Na *Figura 2.29* podemos ver a queda de tensão nas resistências em pormenor e o valor que será lido pelo comparador.

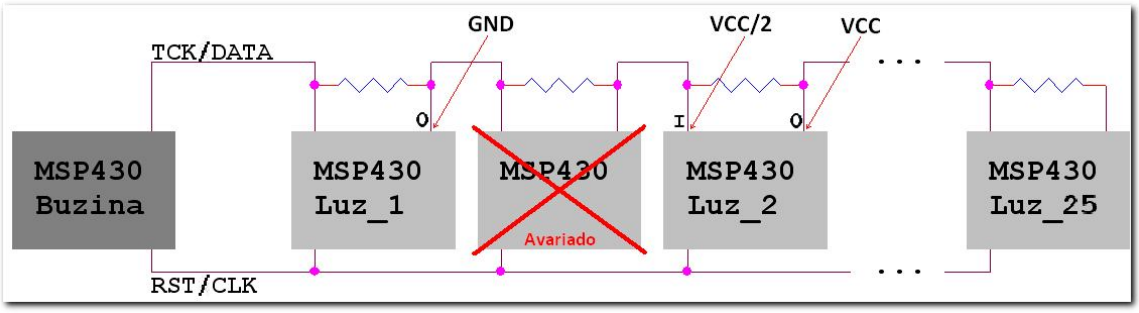


Figura 2.28 - Detecção de avaria em duas luzes consecutivas, com uma avariada entre elas

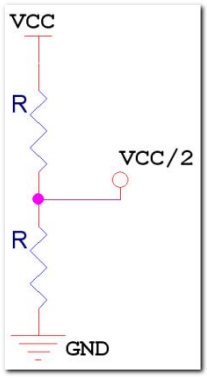


Figura 2.29 - Queda de tensão nas resistências e valor lido no comparador

Este procedimento é depois repetido para todos os microcontroladores.

De salientar que este processo é algo limitado, pois não permite detectar avaria no primeiro ou último microcontrolador (pois não existe microcontrolador à esquerda e à direita destes, respectivamente) e não permite também distinguir entre um ou mais microcontroladores avariados consecutivos (pois o comparador utiliza apenas uma tensão de referência, $VCC/4$, e a tensão aplicada no terminal esquerdo da resistência do microcontrolador com *ID* imediatamente a seguir ao enésimo, será sempre superior a $VCC/4$ e aumenta com o número de microcontroladores avariados: um microcontrolador avariado, teremos $VCC/2$; dois microcontroladores avariados, teremos $2VCC/3$; três microcontroladores avariados, teremos $3VCC/4$; etc). A *Figura 2.30* mostra as quedas de tensão nas resistências para quatro casos possíveis: Sem microcontrolador avariado, para um microcontrolador avariado, para dois microcontroladores consecutivos avariados e para três microcontroladores consecutivos avariados.

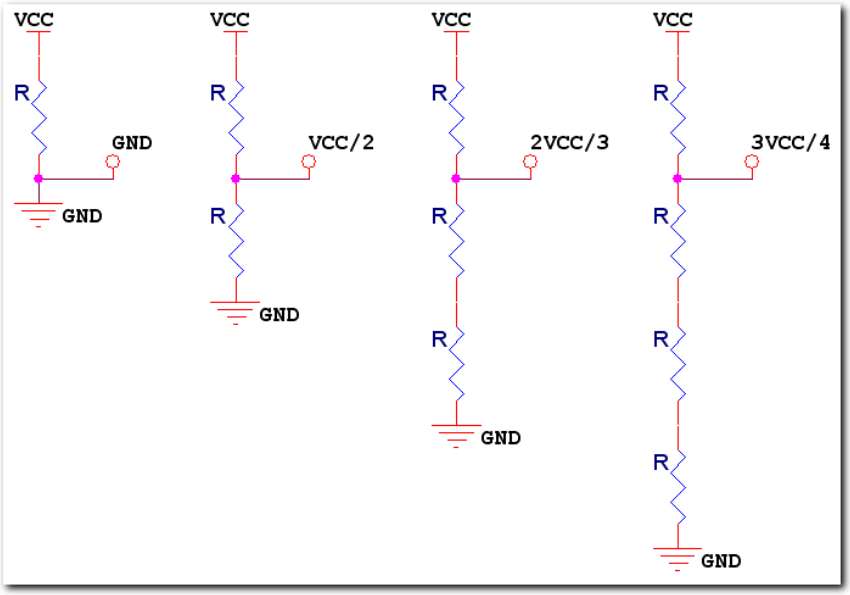


Figura 2.30 - Quedas de tensão nas resistências para quatro casos possíveis

2.6.1 - Alterações efectuadas

Com a finalidade de se armazenar na buzina a informação relativa ao número e posição de cada luz avariada, definiu-se um vector de dimensão igual a 26 onde cada índice desse vector representa o *ID* da respectiva luz. Se determinada posição contiver o valor zero, significa que essa luz está a funcionar, mas se contiver o valor um significa que essa luz está avariada.

Como a o algoritmo de detecção de luzes avariadas não permitia detectar avaria nos microcontroladores da primeira e última luz, foi decidido implementar também um modo de detectar essas avarias. Para o efeito, foram analisadas um conjunto de soluções possíveis.

Uma das soluções passava por colocar um microcontrolador adicional no início e no final da pista de luzes que serviriam apenas para ajudar na detecção de avaria no primeiro e no último microcontrolador.

O microcontrolador adicional no início da pista de luzes seria configurado com papel passivo, enquanto o microcontrolador da primeira luz seria configurado com papel activo e compararia a tensão no seu pino *CAO* com a tensão de referência $VCC/4$.

O microcontrolador da última luz seria depois configurado com papel passivo, enquanto o microcontrolador adicional colocado no final da pista de luzes seria configurado com papel activo e compararia a tensão no seu pino *CAO* com a tensão de referência $VCC/4$. Na *Figura 2.31* está demonstrado o novo esquema com as principais ligações assinaladas.

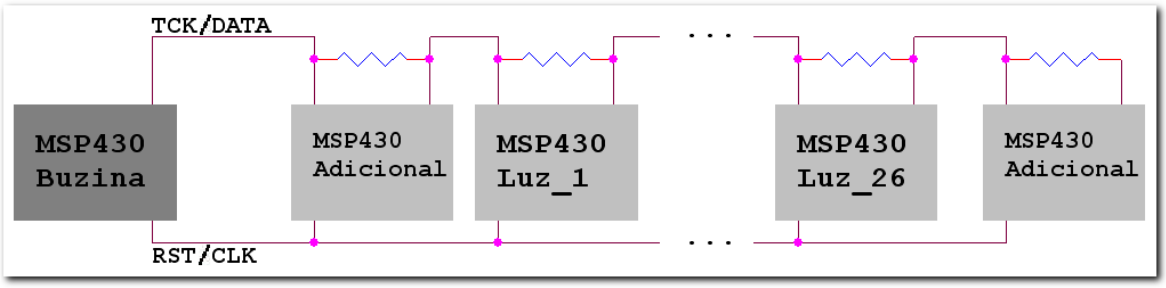


Figura 2.31 - Solução com dois microcontroladores adicionais

Esta solução resultaria na perfeição, mas implicava a necessidade de adicionar mais dois microcontroladores na pista de luzes que serviriam apenas para detectar avaria no microcontrolador da primeira e última luz.

Outra solução passava por diferenciar os microcontroladores da primeira e última luz dos microcontroladores das restantes luzes. Para isso usavam-se dois pinos livres desses microcontroladores e ligavam-se um ao outro, definia-se um pino como *input* e o outro como *output high*. Nos restantes microcontroladores das outras luzes, bastava ligar-se o pino definido como *input* directamente ao *GND*. Agora bastava activar os pinos de *output high* dos microcontroladores da primeira e última luz e fazer a respectiva leitura nos pinos de *input* desses mesmos microcontroladores. Se o valor lido fosse igual a 1, então os microcontroladores estariam a funcionar correctamente, se o valor lido fosse igual a 0, então esse microcontrolador estaria avariado, pois foi um dos outros microcontroladores que foi activado. Na *Figura 2.32* podemos verificar as alterações que seriam necessárias efectuar em cada luz.

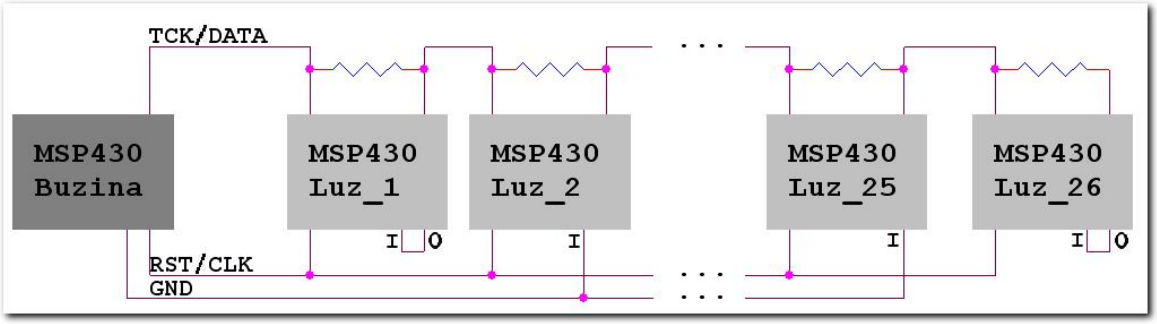


Figura 2.32 - Solução com leitura de valor num determinado pino

Esta solução também resultaria, mas diferenciava os microcontroladores da primeira e da última luz de todos os restantes, implicando a necessidade de haver dois tipos distintos de luzes: um tipo com os pinos ligados entre si para a primeira e última luz, e um tipo com o pino de *input* ligado ao *GND* para todas as restantes luzes.

Uma outra solução encontrada, passava por aproveitar a ligação entre a buzina e a primeira luz de modo a que a buzina tomasse o papel do elemento passivo para a detecção de avaria na primeira luz. Desse modo a buzina colocaria o pino ligado à linha *TCK/DATA* a *output low* e o

microcontrolador da primeira luz colocaria os seus pinos *I/O_R* a *output high*. Se houvesse uma luz avariada, o microcontrolador da primeira luz leria no seu pino *CAO* uma tensão igual a $VCC/2$ e se não houvesse uma luz avariada teria no seu pino *CAO* uma tensão igual a *GND*.

Uma eventual avaria na última luz, seria detectada através de um método de exclusão por partes. Efectuar-se-ia a soma do número total de luzes activas com o número total de luzes avariadas. Se o resultado dessa soma fosse igual ao número total de luzes da pista (ou seja, igual a 26) a última luz não estaria avariada, mas se o resultado fosse igual ao número total de luzes da pista menos um (ou seja, igual a 25) então, por exclusão de partes, a última luz estaria avariada.

Esta solução, embora não levasse a nenhuma alteração nas ligações da buzina e das luzes, era contudo de difícil execução devido ao facto da comunicação entre a buzina e as luzes se efectuar pela mesma linha que seria usada para enviar o sinal a *output low*. Desse modo, seria muito difícil enviar a partir da buzina os comandos de configuração para a primeira luz, enviar depois o sinal a *output low* e receber ainda o resultado do comparador da primeira luz. Uma forma de evitar esse problema, seria enviar o sinal de *output low* por um fio diferente, mas seria então necessário mais um fio de comunicação entre a buzina e as luzes. Como o fio usado na comunicação tem apenas 4 condutores e todos eles estão a ser usados, a solução de usar mais um fio estaria fora de questão.

Outro problema que se levanta é quando existe mais do que um microcontrolador consecutivo avariado, pois neste caso, como não é possível distinguir um microcontrolador avariado de dois, três, ou até mais microcontroladores avariados consecutivos, a soma do número total de luzes activas e de luzes avariadas não daria o número total de luzes da pista mesmo com a última luz a funcionar. Este problema já não se coloca se se encontrar um método eficaz para detectar mais do que uma luz avariada consecutiva.

A *Figura 2.33* representa a situação em que apenas existe uma luz avariada e neste caso o número total de luzes activas é 25. A soma do número de luzes activas com o número de luzes avariadas é igual a 26 e deduz-se então que a última luz não está avariada. A *Figura 2.34* representa uma situação onde existem duas luzes consecutivas avariadas. Como o sistema não é capaz de detectar duas luzes consecutivas avariadas, o sistema interpretará como sendo apenas uma luz avariada. Neste caso teremos 24 luzes activas e uma luz avariada, sendo que a sua soma é igual a 25 e o sistema deduzirá, incorrectamente, que a última luz está avariada quando na realidade não está.

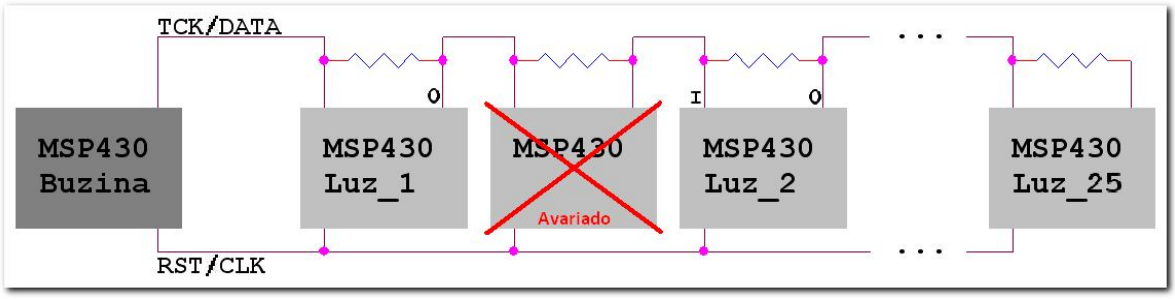


Figura 2.33 - Quando não existe mais do que uma luz consecutiva avariada o sistema funciona correctamente

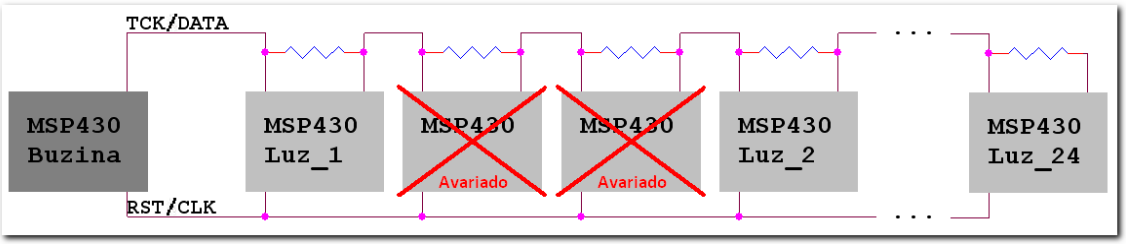


Figura 2.34 - Quando existe mais do que uma luz consecutiva avariada o sistema detecta incorrectamente uma avaria na última luz

Finalmente, a opção seleccionada foi muito semelhante à anterior, mas em vez de se utilizar a buzina como elemento passivo, usa-se como elemento activo na detecção de avaria na primeira luz e usa-se o próprio comparador da buzina e um pino adicional para determinar se a primeira luz está ou não avariada. Foi necessário alterar ligeiramente o circuito da buzina a fim de tornar possível o uso desta solução: foi adicionada uma resistência entre os pinos *P2.4* (que será utilizado como *output high*) e o pino *P3.4* (pino utilizado para a comunicação com as luzes) e ligou-se ainda o pino *P3.4* directamente ao pino *CMP* da buzina a fim de se proceder à comparação da tensão presente nesse pino com a tensão de referência $VCC/4$.

Na *Figura 2.35* podemos visualizar as alterações que tiveram que ser efectuadas na buzina. O pino à esquerda da resistência é o pino que é colocado a *high* (como todos os dispositivos configurados em modo activo), o pino à direita da resistência é o comparador da buzina e é conectado ao pino que transporta o sinal *TCK/DATA*. A detecção de avaria na última luz continua a ser efectuada pelo método de exclusão de partes.

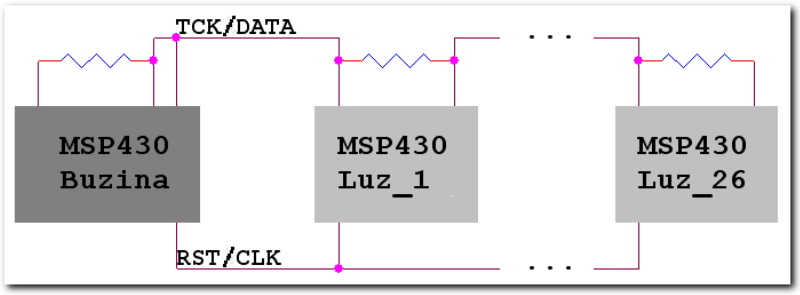


Figura 2.35 - Solução em que a leitura é efectuada pela buzina

Apesar de se assumir que será muito difícil duas luzes consecutivas avariarem ao mesmo tempo, foi procurada uma solução para esta situação.

Como foi já referido, todo o microcontrolador (da família *MSP430x1xx*) possui as duas entradas do seu comparador acessíveis através dos pinos *P2.3* (*CA0*) e *P2.4* (*CA1*). Estas entradas podem também ser ligadas internamente a alguns valores de referência, tais como $VCC/2$ ou $VCC/4$. Tendo isto em análise, e como já estamos a utilizar o pino *CA0* para detectar a existência de uma

luz avariada, iremos agora utilizar o pino *CA1* para detectar a existência de duas luzes consecutivas avariadas.

Antes de mais, será necessário desconectar o pino *P2.4* (*CA1*) dos pinos *I/O_R*. Em sua substituição, podemos ligar o pino *P2.6* que se encontra actualmente desconectado do nosso circuito. Terão ainda que se adicionar duas resistências (*R1* e *R2*) a cada microcontrolador e efectuar as ligações tal como mostrado na *Figura 2.36*. A resistência *R1* estará ligada, do seu lado esquerdo, ao pino *I/O_L*. Do lado direito estará ligada à resistência *R2* e ao pino *CA1* do microcontrolador. A resistência *R2* estará ligada do seu lado direito a um outro pino livre do microcontrolador que denominaremos simplesmente de *I/O* (pode ser o pino *P2.7* que se encontra actualmente desconectado do nosso circuito).

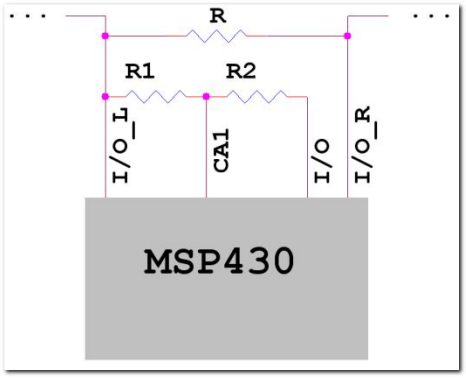


Figura 2.36 - Circuito para detecção de duas luzes avariadas consecutivas

Para que as novas resistências, *R1* e *R2*, não interfiram durante os processos de endereçamento das luzes, procura por uma luz avariada e restante comunicação com a buzina, os pinos *CA1* e *I/O* deverão estar sempre definidos como *input*.

Quando se detectar uma luz avariada através do método normal de detecção, deveremos testar se é apenas um ou se são dois microcontroladores avariados consecutivos. Para começar, colocaremos o pino *I/O_R* do microcontrolador à esquerda do avariado definido como *output high*. De seguida, colocaremos os pinos *I/O_L* e *I/O* do microcontrolador à direita do avariado, definidos como *output low*. Configuraremos então o microcontrolador para efectuar a comparação do valor lido no pino *CA1* com o valor de referência $VCC/4$. No caso de existir apenas um microcontrolador avariado, teremos o circuito equivalente mostrado à esquerda na *Figura 2.37* e o valor lido no pino *CA1* deverá ser superior a $VCC/4$. Se existirem dois microcontroladores avariados, teremos o circuito equivalente mostrado à direita na *Figura 2.37* e o valor lido em *CA1* deverá ser inferior a $VCC/4$.

Em ambos os casos, para termos as aproximações evidenciadas na *Figura 2.37*, devemos sempre considerar que $(R1 + R2) >> R$.

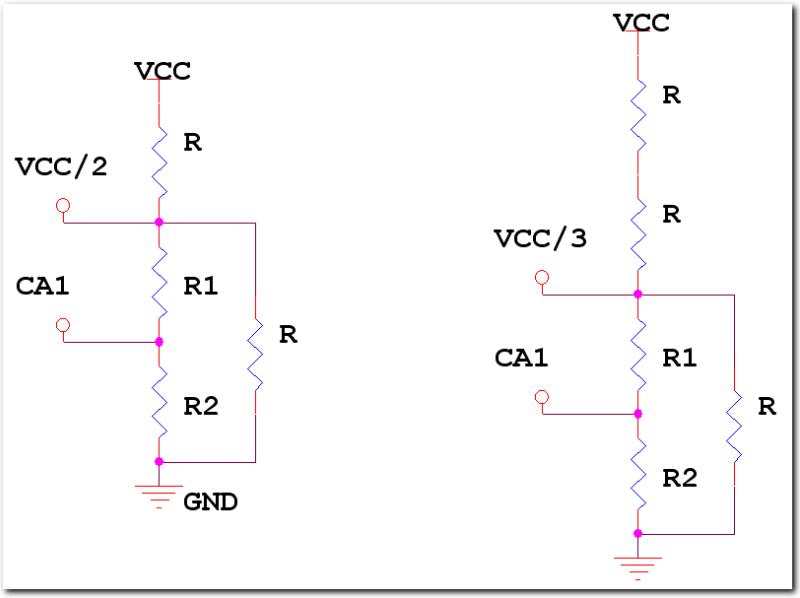


Figura 2.37 - Equivalentes eléctricos para detecção de duas luzes consecutivas avariadas

Este sistema não foi contudo implementado no circuito da pista de luzes, pois eram necessárias efectuar algumas alterações a nível de *hardware* e *software* na mesma. Seria também necessário efectuar algumas alterações a nível de *software* na buzina. O funcionamento do seu equivalente eléctrico foi no entanto testado com sucesso num circuito montado numa placa branca. A sua futura implementação e teste na pista de luzes ficarão definidas para trabalho futuro.

Este método é bastante simples e, acima de tudo, relativamente barato. Estas alterações terão que ser efectuadas em todos os microcontroladores de todas as luzes, o que significa que o custo final da implementação deste sistema terá que ser multiplicado por 26,ou seja, neste caso, tem um custo adicional de 52 resistências.

Capítulo 3. Buzina

3.1 – Introdução

Neste capítulo iremos demonstrar detalhadamente o funcionamento da buzina. Será apresentado todo o trabalho realizado anteriormente e todas as alterações efectuadas durante a realização deste projecto. De realçar que o *software* da buzina disponibilizado no início do projecto também não se encontrava a funcionar.

A buzina é o elo de comunicação entre todos os módulos do *Pacer2*. A sua função principal é receber comandos enviados pela *PDA* e retransmiti-los para as luzes enquanto disponibiliza informações úteis ao utilizador num *display LCD*. A buzina é também a responsável por dar o sinal de partida através de uma sirene nela implementada.

A Buzina é constituída assim por um microcontrolador, um *LCD*, uma sirene e um módulo *wireless*.

A função do microcontrolador da buzina é programar, inicializar e controlar todos os componentes da buzina e inicializar e programar a pista de luzes. O microcontrolador faz o *BSL* da pista de luzes sempre que o sistema é iniciado, recebe e interpreta comandos enviados pela *PDA*, envia informações e *acknowledges* de volta para a *PDA*, envia informações para o *display LCD*, envia os comandos recebidos por *wireless* para a pista de luzes, recebe e interpreta informações enviadas pela pista de luzes e liga ou desliga a buzina.

Pretende-se ainda que a taxa de comunicação entre o microcontrolador e o módulo *wireless* seja de 9.6*kbaud*.

O envio e a recepção de informação por parte da buzina utilizará as regras do *RS232*, onde enviamos um *low start bit* e um *high stop bit* a delimitar cada *byte* de dados.

3.2 - Protocolo wireless

O módulo *wireless* permite três tipos de codificação: *NRZ (Non-Return-to-Zero)*, *Manchester* e *Transparent UART*. A codificação que será utilizada, e a recomendada pelo fabricante, é a codificação *Manchester*. Como a codificação *Manchester* utiliza dois símbolos para transmitir cada *bit*, será então necessário configurar os módulos para utilizarem uma frequência de comunicação de 19.2*kbaud* a fim de se obter uma taxa de transferência de 9.6*kbaud* como pretendido. Um exemplo de codificação *Manchester* pode ser verificado na *Figura 3.1*.

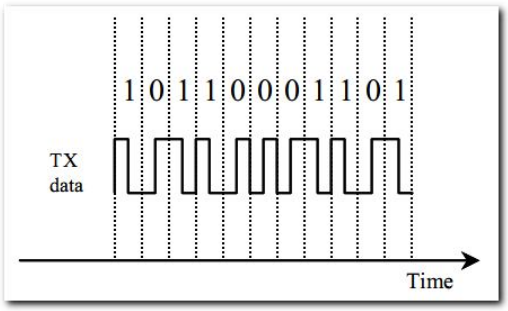


Figura 3.1 - Codificação Manchester[48]

O sinal de *clock* presente no pino *DCLK* será assim automaticamente configurado pelo módulo *wireless* para uma frequência de 9.6*kHz*. Na *Figura 3.2* podemos visualizar um exemplo da leitura de um *byte* e respectivos *start* e *stop bit* por parte do *CC1000*.

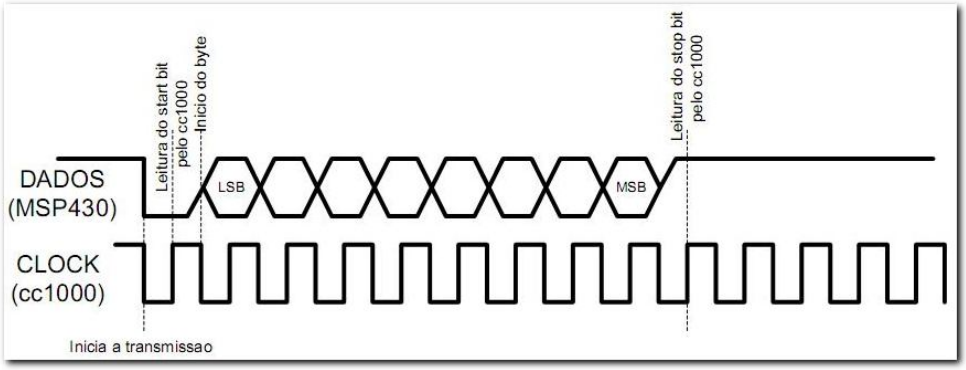


Figura 3.2 - Comunicação RS232[49]

[48] Imagem retirada de: [Texas Instruments, 2009]

3.3 - *LCD*

O *LCD* contém um controlador próprio. Para escrever no *LCD* precisamos apenas de configurar o seu controlador segundo um protocolo específico deste. Na *Figura 3.3* podemos ver que a interface com o microcontrolador da buzina é feita através de um barramento de dados paralelo de 8*bits* e três sinais de controlo, *E* (*enable*), *R/W* (*read/write*) e *RS* (*data/command*).

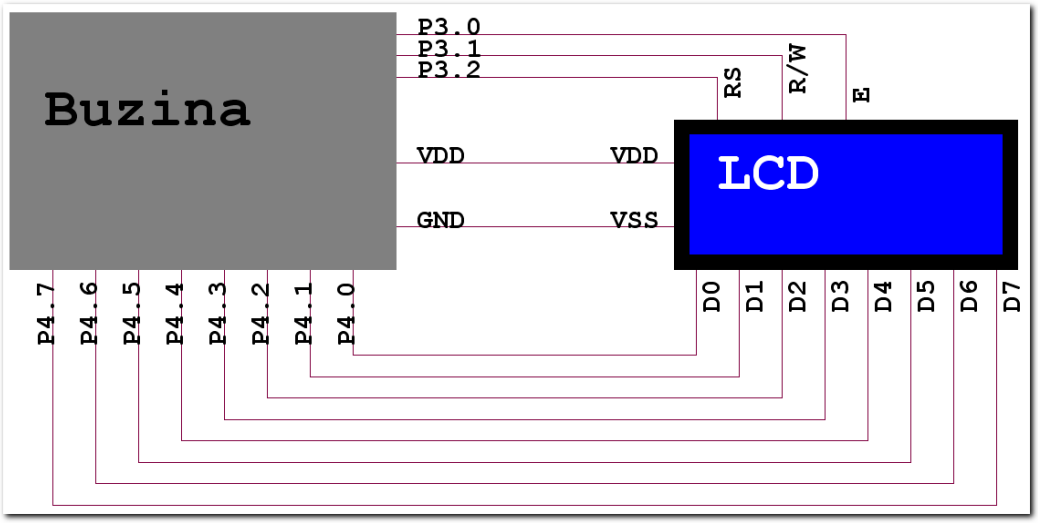


Figura 3.3 - Diagrama de ligações entre o microcontrolador da buzina e o LCD

O *LCD* dispõe ainda de um pino de controlo de contraste do *LED* azul de fundo. O contraste pode assim ser controlado por um potenciómetro colocado entre o pino *VCI* do *LCD* e *VDD*, tal como mostra a *Figura 3.4*.

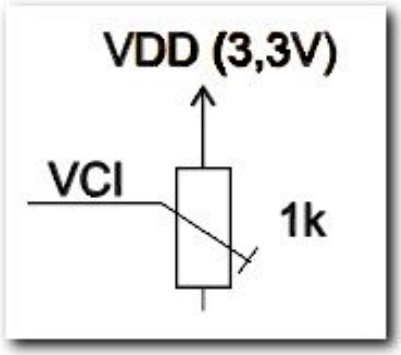


Figura 3.4 - Regulador de intensidade luminosa[50]

---

[49] Imagem retirada de: [Cabrita, 2007]
[50] Imagem retirada de: [Electronic Assembly, 2006]

Funções para escrever caracteres no *display LCD*, escrever valores inteiros em base de dez, colocar símbolos, mover o cursor para qualquer posição e limpar o ecrã, tinham já sido desenvolvidas e testadas.

3.3.1 - Alterações efectuadas

A fim de confirmar o seu correcto funcionamento, todas estas funções foram novamente testadas uma a uma e todas elas estavam a funcionar correctamente. Não foi portanto necessário fazer qualquer alteração ao seu código, apenas foi correctamente comentado a fim de facilitar futuras correcções ou alterações.

3.4 - Sirene

A sirene será controlada por um pino *I/O* do microcontrolador, configurado para funcionar sempre como *output*. A sirene apitará sempre que o pino esteja a *high* e permanecerá desligada enquanto o pino estiver a *low*. Na *Figura 3.5* podemos ver o esquema da ligação entre o *MSP* e a sirene.

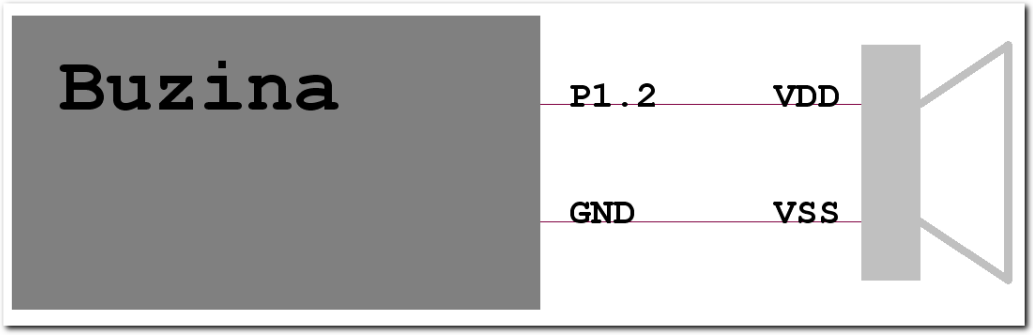


Figura 3.5 - Esquema de ligações entre a buzina e a sirene

3.4.1 - Alterações efectuadas

Não existia, no *software* previamente disponibilizado, nenhuma função que controlasse e interagisse com a sirene. Foi então criada uma função que aceita três parâmetros: o primeiro corresponde ao tempo total (em milissegundos) de funcionamento da sirene, o segundo será o intervalo de tempo (em milissegundos) em que esta estará ligada e o terceiro o intervalo de tempo (em milissegundos) em que estará desligada. Quer isto dizer que esta função permite repetições sonoras da sirene durante um intervalo definido de tempo, por exemplo:

- Primeiro parâmetro corresponde a 4000*ms*;
- Segundo parâmetro corresponde a 800*ms*;

- Terceiro parâmetro corresponde a 200*ms*.

Então a sirene ligar-se-á durante 800*ms*, desligar-se-á durante 200*ms* e tudo isto repete-se durante 4000*ms*.

Todo o código referente à sirene foi assim comentado, tendo em vista facilitar uma futura análise e alteração do respectivo código.

3.5 - Pista de Luzes

A buzina comunica com a pista de luzes através de dois pinos genéricos de *I/O* do microcontrolador.

O primeiro pino é o *P3.4/UTXD0* e é utilizado para enviar os dados para as luzes, alterar o estado do pino *TCK* do microcontrolador de cada luz e enviar os dados em modo *BSL*. Para enviar os dados por *BSL*, é utilizado o protocolo *UART0* como explicado anteriormente.

O segundo pino é o *P1.0/TACLK* e é utilizado para alterar o estado do pino *RST/NMI* e é o pino utilizado para enviar o sinal de *clock* para a transferência de dados com as luzes. Esta transferência ocorre, provocando uma *NMI* no microcontrolador de cada luz. Cada luz tem a mesma rotina de atendimento à *NMI* que procede à leitura/escrita de um *bit* de cada vez.

Na *Figura 3.6* podemos verificar que a alimentação da pista de luzes é fornecida pela buzina através de um *MOSFET* (*Metal-Oxide-Semiconductor Field-Effect Transistor*) tipo *p* funcionando como chave. Esta configuração permite assim desligar a alimentação da pista de luzes sempre que esta não esteja em funcionamento. Este *MOSFET* é também ele essencial para efectuar um *BSL* aos microcontroladores das luzes depois de estas terem sido completamente inicializadas e configuradas pela primeira vez durante uma sessão, pois assim que tal aconteça, o pino *RST/NMI* responsável, conjuntamente com o pino *TCK*, pela sequência de inicialização do *BSL* em cada luz, estará configurado para efectuar uma *NMI* aquando do flanco positivo do sinal de *clock* a ele ligado, o que impossibilita por completo a inicialização da sequência de entrada em modo *BSL*. Para solucionar este problema, o *MOSFET* permite desligar a pista de luzes, funcionando assim como um *reset* da pista de luzes controlado pelo *software* da buzina.

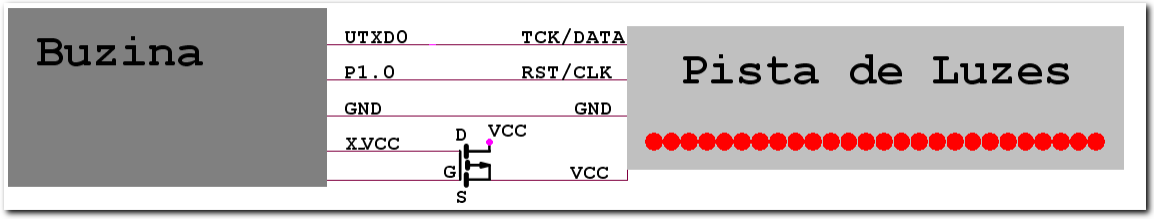


Figura 3.6 - Esquema de ligações entre a buzina e a pista de luzes

Verificou-se também que a intensidade luminosa na última luz é mais baixa do que a intensidade luminosa da primeira. Esta situação não é de todo desejável, pois corre-se o risco de para as situações em que seja necessário que as últimas luzes tenham uma intensidade elevada, as primeiras luzes fiquem com uma intensidade tão grande que possa ferir a visão dos atletas. Do mesmo modo, se se pretende que as primeiras luzes tenham uma intensidade relativamente baixa, corre-se o risco das últimas luzes ficarem com uma intensidade tão baixa que deixem de ser perceptíveis. É precisamente para uniformizar esta intensidade luminosa que existe o comando de dimensionamento de *PWM*.

Do mesmo modo, verificou-se que a cor vermelha apresenta uma intensidade luminosa maior que as restantes cores, e é para eliminar esta diferença de intensidades luminosas que existe o comando dimensionamento de luminescência.

3.5.1 - Alterações efectuadas

Como foi mencionado no segundo capítulo, foi necessário adicionar mais duas ligações ao microcontrolador da buzina a fim de possibilitar a detecção de avaria no funcionamento da primeira luz. Essas ligações podem ser observadas na *Figura 3.7*, onde adicionamos uma resistência entre os pinos *P2.4* e *CMP* e ligamos este último ao pino *UTXD0*. Essa resistência é necessária para criar uma determinada queda de tensão entre esses dois pinos, a fim de efectuar uma comparação do valor lido no pino *CMP* com o valor de referência $VCC/4$.

De seguida, foi necessário criar uma rotina no *software* da buzina que analise o vector responsável por armazenar o *ID* das luzes avariadas e compense o *ID* das luzes seguintes, sempre que seja detectada alguma avaria.

O *ID* é compensado transmitindo o comando de atribuição manual de *ID* com o valor seguinte ao *ID* actual da luz a alterar. É muito importante que essa alteração comece a ser efectuada da última luz para trás, pois só assim se garante que as luzes fiquem com o *ID* correcto. Se tal fosse realizado da luz seguinte à avariada para a frente, o resultado seria desastroso, pois todas as luzes seguintes à avariada ficariam com o mesmo *ID* e este seria igual ao *ID* da última luz.

Para verificar o bom funcionamento da compensação dos *ID* foi criada uma função que acende e apaga todas as luzes por ordem ascendente e depois descendente de segundo em segundo. Assim se consegue ver se as luzes se identificam com o *ID* correcto, mesmo que existam luzes avariadas entre elas.

De seguida analisamos a ligação do *MOSFET* que controla a alimentação da pista de luzes e verificamos que este encontrava-se com os pinos de dreno e fonte trocados, visto tratar-se um *MOSFET* do tipo *p*.

Na *Figura 3.7* podemos ver o esquema completo das ligações entre a buzina e a pista de luzes, onde podemos observar as novas ligações adicionadas para possibilitar a detecção de avaria na primeira luz e a troca do dreno pela fonte no *MOSFET* responsável pela alimentação da pista de

luzes. De referir ainda que o pino *X_VCC* é o pino que controla a tensão da porta do *MOSFET* e que corresponde ao pino *P1.1* do microcontrolador da buzina.

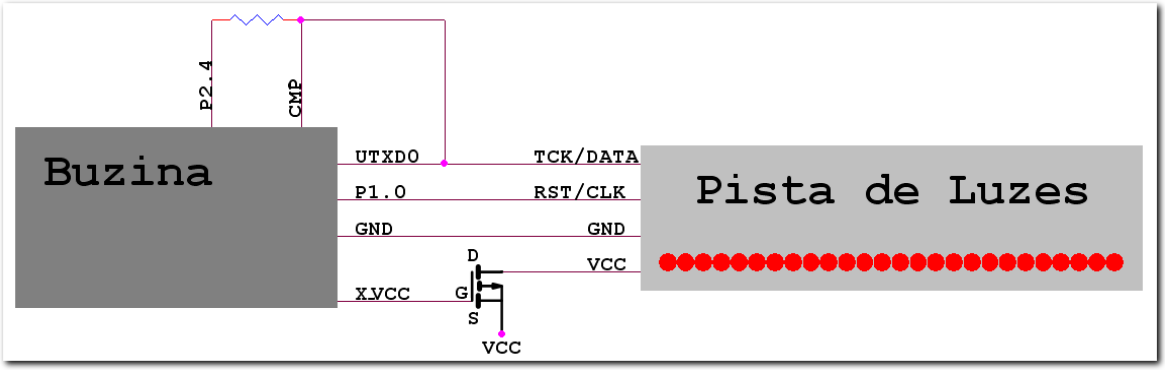


Figura 3.7 - Esquema de ligações final entre a buzina e a pista de luzes

3.6 - Firmware updates através de BSL

Um dos objectivos principais deste trabalho é permitir que possam ser efectuados *firmware* updates tanto nas luzes, como na buzina. Já aqui foi visto como se efectuam os *firmware updates* das luzes com a ajuda da buzina, falta agora discutir como serão efectuados os *firmware updates* da própria buzina.

A solução encontrada passa pela ligação de um microcontrolador adicional na buzina, controlador esse que terá como único objectivo a programação da buzina com um novo *firmware* sempre que este se encontre disponível. Esse *firmware* estará contido num ficheiro que será descarregado para a *PDA* e enviado depois por *wireless* para o reprogramador da buzina que fará com que esta entre em modo *BSL* e o reprograme.

Para que tal seja possível, é necessário que, para evitar erros de programação e curto-circuitos e independentemente do método escolhido para ligar o reprogramador da buzina, pelo menos o pino *P2.2* da buzina esteja livre, pois é através deste pino que os dados serão recebidos por *BSL*. O pino *P1.1* não necessita de ser reservado para a transmissão dos *acknowledges* do *BSL*, pois já verificamos na pista de luzes que o seu correcto funcionamento é possível mesmo sem a recepção por parte do transmissor do respectivo *acknowledge* após o envio de um comando. Teremos apenas que ter em conta que esse pino continuará mesmo assim a transmitir o sinal de *acknowledge* quando recebe um comando por *BSL* e será assim necessário garantir que este pino não estará activo e a efectuar uma outra acção enquanto o *BSL* estiver a decorrer.

3.7 - Módulo *wireless*

É através do módulo wireless que a buzina comunica com a *PDA*. Sempre que a buzina é ligada, é necessário programar o *CC1000* para que este funcione com os parâmetros adequados (frequência de transmissão, *baud rate*, modo receptor ou transmissor, etc).

A ligação entre o microcontrolador e o *CC1000* faz-se com o auxílio de 7 pinos de *I/O* como mostrado na *Figura 3.8*.

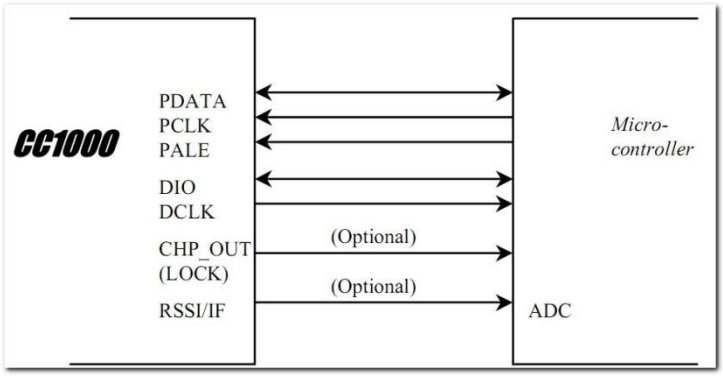


***Figura 3.8 - Interface de comunicação CC1000-microcontrolador*[51]**

Como o *CC1000* está já incorporado no módulo *MMcc1000*, a *Figura 3.9* mostra-nos a interface deste com o microcontrolador. Nesta figura o pino *CHP* do módulo *MMcc1000* não está ligado ao microcontrolador, mas na realidade ele está ligado a um pino genérico de *I/O*.

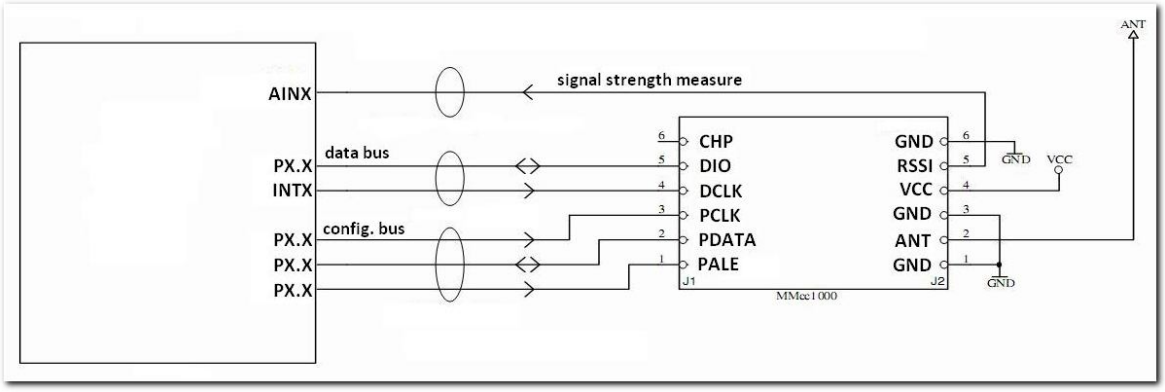


***Figura 3.9 - Interface de comunicação MMcc1000-microcontrolador*[52]**

Os pinos *PDATA* (dados), *PCLK* (sinal de *clock*) e *PALE* (sinaliza se os bits enviados são dados ou comandos) são utilizados apenas para programar o *CC1000*. Os outros quatro pinos são utilizados

[51] Imagem retirada de: [Texas Instruments, 2009]
[52] Imagem retirada de: [Propox, 2009]

para as comunicações: *DIO* (*data input/output*), *DCLK* (sinal de *clock*), *RSSI* (medição da força do sinal) e *CHP_OUT* (sinal de sinalização).

A configuração é efectuada através de um endereço de 7*bits*, 1*bit* indicador de operação de leitura/escrita (*high* para escrita e *low* para leitura) e 1*byte* de dados.

Os dados são escritos nos registos do *CC1000* no flanco descendente do sinal *PCLK*. Inicialmente com o sinal *PALE* a *low*, é enviado o primeiro *byte* contendo o endereço e o *bit* de leitura/escrita, sendo que o primeiro *bit* enviado é o *bit* mais significativo do endereço e o *bit* de leitura/escrita é o último enviado. De seguida o sinal *PALE* passa a *high* e é enviado o *byte* de dados, sendo que o primeiro *bit* enviado é o mais significativo.

A leitura de um registo deve ser efectuada quando o sinal *PCLK* está a *low*, pois segundo o *datasheet* do dispositivo, o controlador do *CC1000* altera os dados no flanco ascendente de *PCLK*. As *Figura 3.10* e *Figura 3.11* mostram as operações de escrita e de leitura, respectivamente.

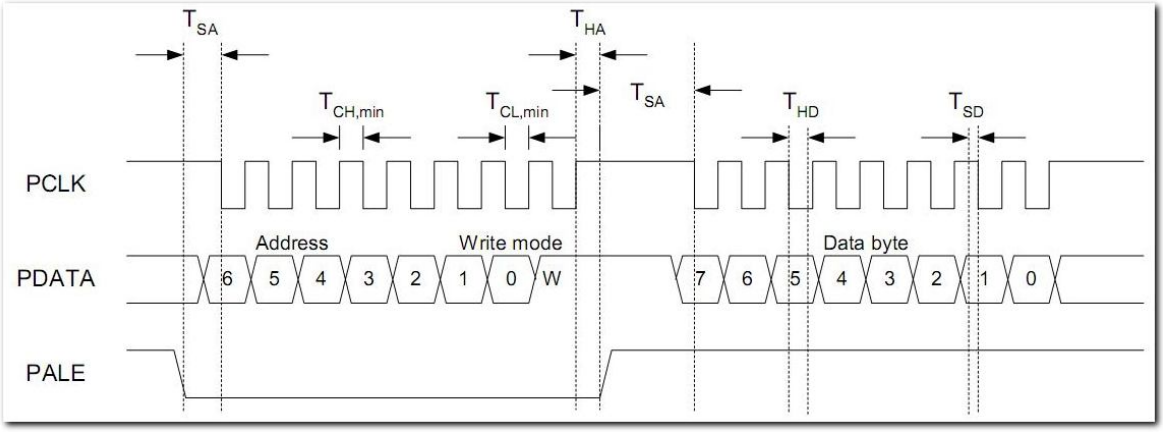


Figura 3.10 - Processo de escrita num registo do CC1000[53]

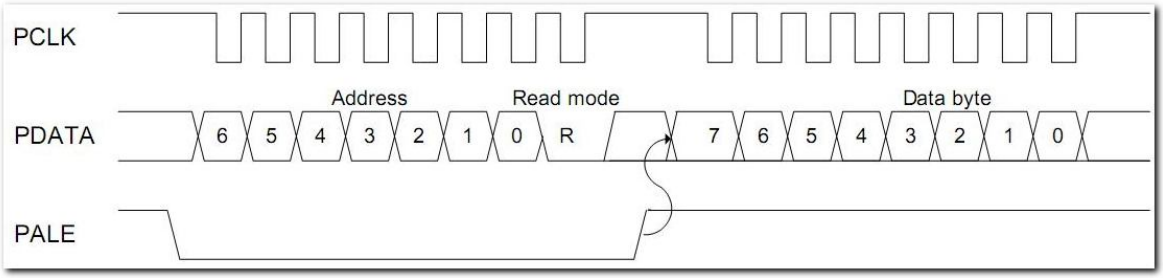


Figura 3.11 - Processo de leitura de um registo do CC1000[54]

[53] Imagem retirada de: [Texas Instruments, 2009]
[54] Imagem retirada de: [Texas Instruments, 2009]

A transferência de dados realiza-se através de 2*bytes*, o primeiro contém o endereço e o *bit* de leitura/escrita e o segundo contém os *bytes* a enviar/receber.

O envio e a recepção de informação entre os vários constituintes da buzina utilizará como protocolo de comunicação o modo *UART1* com regras do *RS232*, onde enviamos um *low start bit* e um *high stop bit* a delimitar cada *byte* de dados.

O microcontrolador da Buzina transmitirá dados para o módulo *wireless* através do pino *UTXD1* que é o pino utilizado para transmitir informação através do módulo *UART1* do *MSP430F1611* e receberá dados através do pino *URXD1* que é o pino utilizado para receber informação através do módulo *UART1* do *MSP430F1611*.

Como o módulo *wireless* dispõe de um pino único para entrada e saída dos dados, *DIO*, e como o modo *UART* dispõe de dois pinos independentes para entrada e saída dos mesmos dados, *URXD1* e *UTXD1* respectivamente, será necessário conectar o pino *DIO* simultaneamente aos pinos *URXD1* e *UTXD1*. Assim sendo, é necessário ter muito cuidado para evitar curto-circuitos, ou seja, enquanto se recebem dados do pino *DIO* é necessário que o pino *UTXD1* esteja desactivado.

O microcontrolador utilizará um *clock* interno para sincronizar com o sinal de 9.6*kHz* utilizado na transferência de dados. Esse *clock* interno será o *SMCLK*[55] modulado a 9.6*KHz*. Desse modo, não será necessário conectar o pino *DCLK* (pino onde está acessível o sinal de *clock* interno do módulo *wireless*) ao microcontrolador.

O esquema simplificado da ligação entre a buzina e o módulo *wireless* pode ser analisado na *Figura 3.12*.

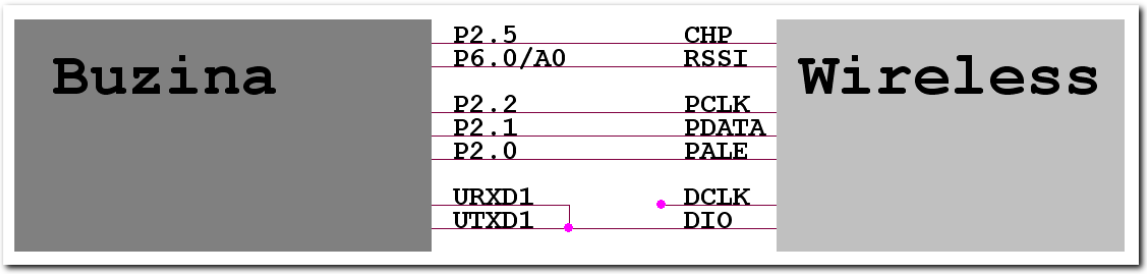


Figura 3.12 - Esquema de ligações entre a buzina e o módulo wireless

3.7.1 - Alterações efectuadas

Como mencionado anteriormente, é necessário que o pino *P2.2* da buzina esteja livre para permitir futuros *firmware updates*. Este pino estava ligado ao pino *PCLK* do módulo *wireless* e era responsável por fornecer o sinal de *clock* durante a sua configuração. Foi necessário então alterar

[55] 8MHZ

a ligação e decidimos ligá-lo ao pino *P2.6* que se encontrava livre, tal como se pode visualizar na *Figura 3.13*.

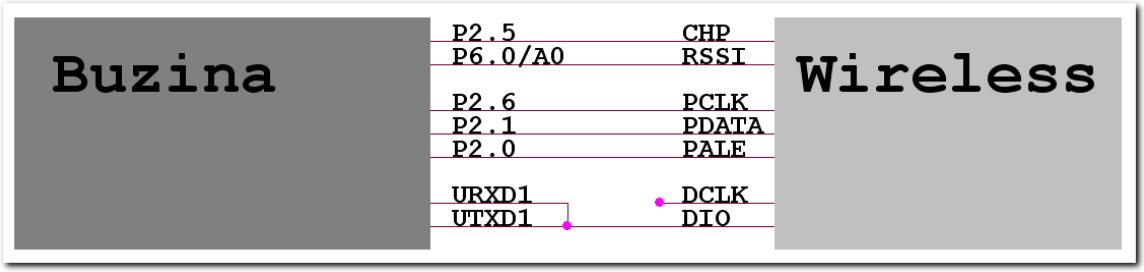


Figura 3.13 - Esquema de ligações entre a buzina e o módulo wireless

A comunicação *wireless* não pôde no entanto ser ainda testada, pois falta corrigir o código do módulo *wireless* presente no simulador de *PDA*. Apenas depois dessas correcções e com os dois módulos em funcionamento (um configurado em modo transmissor e o outro configurado em modo receptor) é que será possível verificar se o restante código está a funcionar.

No final, todo o código referente ao módulo da buzina foi devidamente comentado para facilitar uma futura análise e alteração do mesmo.

3.8 - *Real Time Clock*

Para sincronizar o funcionamento de todos os módulos e do envio e execução de cada comando, foi necessário implementarmos um *RTC (Real Time Clock)*. Este *RTC* será controlado por um cristal de 8*MHz* de frequência e apresenta uma resolução de 500μs.

O *RTC* foi então configurado no modo *up* de contagem, onde o *timer* do microcontrolador conta repetidamente de 0 até um valor definido (neste caso, para ter uma resolução de 500μs com um cristal de 8*MHz*, esse valor será definido como 4000[56]). Ao atingir esse valor, o *timer* recomeça a contagem a partir de 0 e incrementa uma variável definida como *ctime*. Cada incremento desta variável corresponde a 500μs.

Como será este relógio que permitirá que a buzina execute as operações no tempo certo, o controlo por parte da *PDA* não será feito *online*, mas sim através de comandos que executarão passado algum tempo bem definido. Esses comandos serão enviados pela *PDA* e armazenados na buzina e cada comando terá um tempo de execução a ele associado. Quando o *RTC* da buzina atingir esse tempo, o comando é então executado.

[56] $(500x10^{-6})^{-1}=2000$; $8x10^{6}/2000=4000$;

Capítulo 4. Simulador de PDA

4.1 – Introdução

Neste capítulo iremos demonstrar detalhadamente o funcionamento do simulador de *PDA*. Será apresentado todo o trabalho realizado anteriormente e todas as alterações efectuadas durante a realização deste projecto. De realçar que o *software* do simulador de *PDA* disponibilizado no início do projecto também não se encontrava a funcionar.

O simulador de *PDA* é um módulo constituído por um *transceiver wireless* que permitirá comunicar com a buzina através de *RF*, uma *interface RS232* para conectar no futuro a uma verdadeira *PDA* e um microcontrolador *MSP430* que substituirá inicialmente a *PDA*. Inicialmente, o microcontrolador terá na sua memória alguns comandos que serão enviados por *RF* para a buzina através do módulo *wireless*. Quando o sistema estiver a funcionar correctamente com todos os módulos em conjunto, será então ligada a *PDA* ao módulo do simulador de *PDA* e passará a ser esta quem envia os comandos para serem transmitidos por *RF*. O microcontrolador tomará então a única função de programar e inicializar correctamente o módulo *wireless* e retransmitir para este os comandos que a *PDA* lhe enviar de modo a que sejam então recebidos pela buzina.

Estes comandos e informações serão, numa fase inicial, enviados directamente a partir do microcontrolador presente neste, simulando assim uma *PDA*. Numa fase seguinte (terceira tarefa dos objectivos) será uma *PDA*, que estará ligada por *RS232* ao microcontrolador, quem fornecerá os comandos e informações a enviar. Neste caso, o microcontrolador do simulador terá apenas

software necessário para inicializar e configurar o módulo *wireless* e para receber dados da *PDA* e reenviá-los por *RF* para a buzina e vice-versa.

4.2 - Módulo *wireless*

É através do módulo *wireless* que o simulador de *PDA* comunicará com a buzina. Sempre que o simulador de *PDA* é ligado, é necessário programar o *CC1000* para que este funcione com os parâmetros adequados (frequência de transmissão, *baud rate*, modo receptor ou transmissor, etc).

A ligação entre o microcontrolador e o *CC1000* faz-se com o auxílio de 7 pinos de *I/O* tal como referido no capítulo da buzina. Mais uma vez, o envio e a recepção de informação entre os vários constituintes do simulador de *PDA* utilizará como protocolo de comunicação por interrupções com regras do *RS232*, onde enviamos um *low start bit* e um *high stop bit* a delimitar cada *byte* de dados.

Neste caso, será utilizado o sinal de *clock* disponível no pino *DCLK* do módulo *wireless* para originar uma interrupção com o objectivo de efectuar a recepção ou envio de um *bit* de cada vez. O pino *DIO* do módulo *wireless* estará ligado ao pino *P2.4* do microcontrolador e este estará configurado como *input* (e o pino *DIO* como *output*) se estivermos a receber dados, ou como *output* (e o pino *DIO* como *input*) se estivermos a transmitir dados. *Figura 4.1* mostra-nos a interface de ligação entre o módulo *wireless* e o microcontrolador.

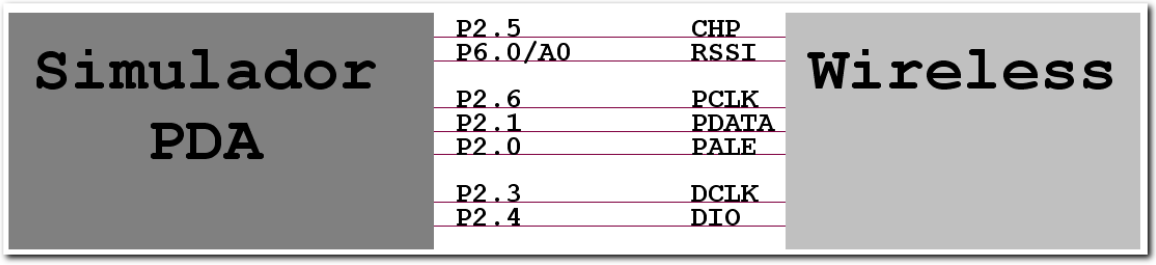


Figura 4.1 - Interface de ligação entre o microcontrolador e o módulo wireless do simulador de PDA

A *Figura 4.2* mostra-nos o diagrama da rotina de atendimento à interrupção que efectua a recepção ou transmissão de um *bit* de dados. Os *buffers RxBuffer* e *TxBuffer* são *buffers* onde está contido o *byte* acabado de receber ou a enviar, respectivamente.

De salientar ainda que é necessário recorrer a um *buffer* adicional do tipo *fifo* para armazenar os *bytes* recebidos, pois o conteúdo do *RxBuffer* está em constante alteração e corríamos o risco de quando acedêssemos ao conteúdo do *RxBuffer* para ler a o *byte* recebido, este já tivesse sido alterado. É através deste *buffer* que sabemos também se um novo *byte* foi recebido ou não: existem dois apontadores para o *buffer*, um aponta para o último *byte* lido e outro para o último

byte recebido; quando estes apontam para *bytes* diferentes, quer dizer que ainda existem *bytes* para ler do *buffer*.

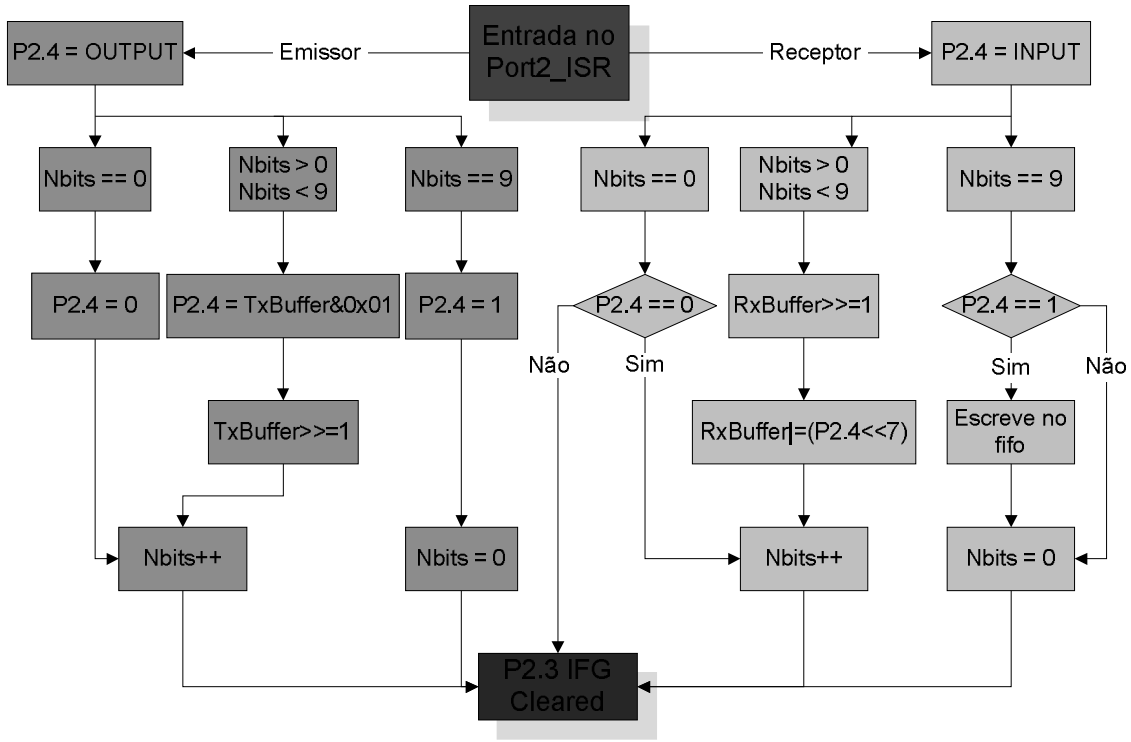


Figura 4.2 – Diagrama da rotina de atendimento à interrupção no pino P2.3

4.3 - Pacotes *wireless*

Com a comunicação por *wireless* a funcionar, podemos começar finalmente a trocar pacotes de dados entre os módulos. Para tal foi necessário recorrer ao código já existente e tentar compreender o que já tinha sido então estabelecido.

4.3.1 - *Data frame* dos pacotes *wireless*

Para receber os pacotes, a buzina está dotada de um vector capaz de armazenar até um máximo de 200*bytes*. Desta forma, a estrutura do *data frame* dos pacotes *wireless* será como a apresentada na *Figura 4.3*.

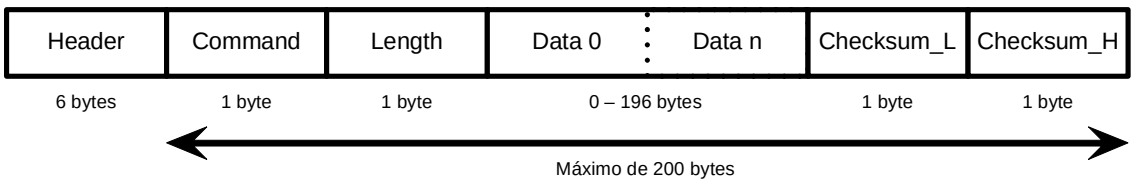


Figura 4.3 - Wireless Data Frame

O campo *Header* é composto por 6*bytes* e serve para identificar os pacotes de dados no meio do ruído recebido pelo receptor. Os 6*bytes* que constituem o *Header* são os seguintes: 0*x*80, 0*x*70, 0*x*61, 0*x*63, 0*x*65 e 0*x*72. Os últimos 5*bytes* correspondem ao código *ASCII* (*American Standard Code for Information Interchange*) das letras *'p'*, *'a*', *'c'*, *'e'*, *'r'*.

O segundo campo é o *Command*, e tal como o seu nome indica, identifica o comando a ser executado. Mais adiante falaremos dos diferentes tipos de comandos existentes.

O comando *Length* indica o tamanho total do pacote desde o *byte Command* até ao último *byte* do *Checksum*. Este campo terá no máximo o valor 200 que é o comprimento do vector presente na buzina para armazenamento dos dados recebidos.

Os *bytes* seguintes são os *bytes* de dados propriamente ditos. Estes podem ser no máximo 196 e no mínimo terá que ser 0, para os casos em que não existem dados. Para se saber o número de pacotes de dados, basta subtrair 4 ao valor lido no campo *Length*.

Os dois últimos *bytes* correspondem ao *Checksum*. O método utilizado para o cálculo do *Checksum* será também ele explicado mais adiante neste capítulo.

4.3.2 - *Byte-Alignment*

Para garantirmos que os *bytes* são correctamente recebidos em sistemas que comunicam através do módulo *UART*, é necessário que o *start bit* de cada *byte* seja reconhecido como tal. Para tal é necessário que imediatamente antes de receber qualquer *byte* de dados o sistema esteja a contar que o próximo *bit* recebido seja o *start bit* do primeiro *byte* de dados. Como o segundo *byte* é enviado imediatamente a seguir ao primeiro *byte*, e assim sucessivamente, se o primeiro estiver correctamente alinhado, garantimos que os seguintes também estarão. Então basta garantirmos que o primeiro *byte* seja bem recebido para que os restantes, que foram enviados sucessiva e imediatamente a seguir uns aos outros, também o sejam.

Uma das formas de garantirmos então o *byte-alignment*, é através do envio, por parte do transmissor, de alguns *bytes* consecutivos imediatamente antes do envio do *Header*, esses *bytes* são os seguintes: 0*x*55, 0*x*44, 0*x*11, 0*x*45, 0*x*15. A recepção destes *bytes*, por parte do receptor, imediatamente antes da recepção do *Header* garante assim o *byte-alignment* para sistemas com as seguintes condições:

- Nunca devem ser transmitidos mais do que 3 zeros ou uns consecutivos para garantir que o dispositivo *RF* esteja relativamente balanceado[57];

[57] Alguns dispositivos RF não conseguem manusear correctamente vários zeros ou uns consecutivos

- É assumido que a comunicação seja estabelecida segundo as convenções do modo *standart UART*, ou seja, envio de um *low start bit*, um *high stop bit* e os *bits* menos significativos são transmitidos primeiro;
- Assume-se que o módulo *UART* receptor reconhecerá um *framing error* e tentará imediatamente a ressincronização assim que detecte um zero onde deveria estar o *high stop bit*;
- Assim que detecte um *framing error*, o módulo *UART* receptor procura primeiro por um *high bit* e só depois procura por um *low bit* e assume que este seja o novo *start bit*.

Com o envio do primeiro *byte* (*0x55*), garantimos que o módulo *UART* receptor esteja num dos seguintes casos:

- Caso A – O módulo *UART* está correctamente sincronizado;
- Caso B – O módulo *UART* está deslocado em 2*bits* (bit 6 está a ser considerado o stop *bit*, *bit* 7 está a ser considerado o *start bit*);
- Caso C – O módulo *UART* está deslocado em 4*bits* (*bit* 4 está a ser considerado o *stop bit*, *bit* 5 está a ser considerado o *start bit*);
- Caso D – O módulo *UART* está deslocado em 6*bits* (*bit* 2 está a ser considerado o *stop bit*, *bit* 3 está a ser considerado o *start bit*);
- Caso E – O módulo *UART* está deslocado em 8*bits* (*bit* 0 está a ser considerado o *stop bit*, *bit* 1 está a ser considerado o *start bit*);

Na *Figura 4.4* podemos ver a evolução da sincronização no receptor, para todos os casos aqui referidos. No final, todos eles deverão estar correctamente alinhados (ou seja, deverão convergir para o caso A).

De realçar que todos os *bytes* de um pacote enviados pelo módulo *wireless* serão recebidos sequencialmente pelo receptor, sem qualquer *bit* de ruído entre eles.

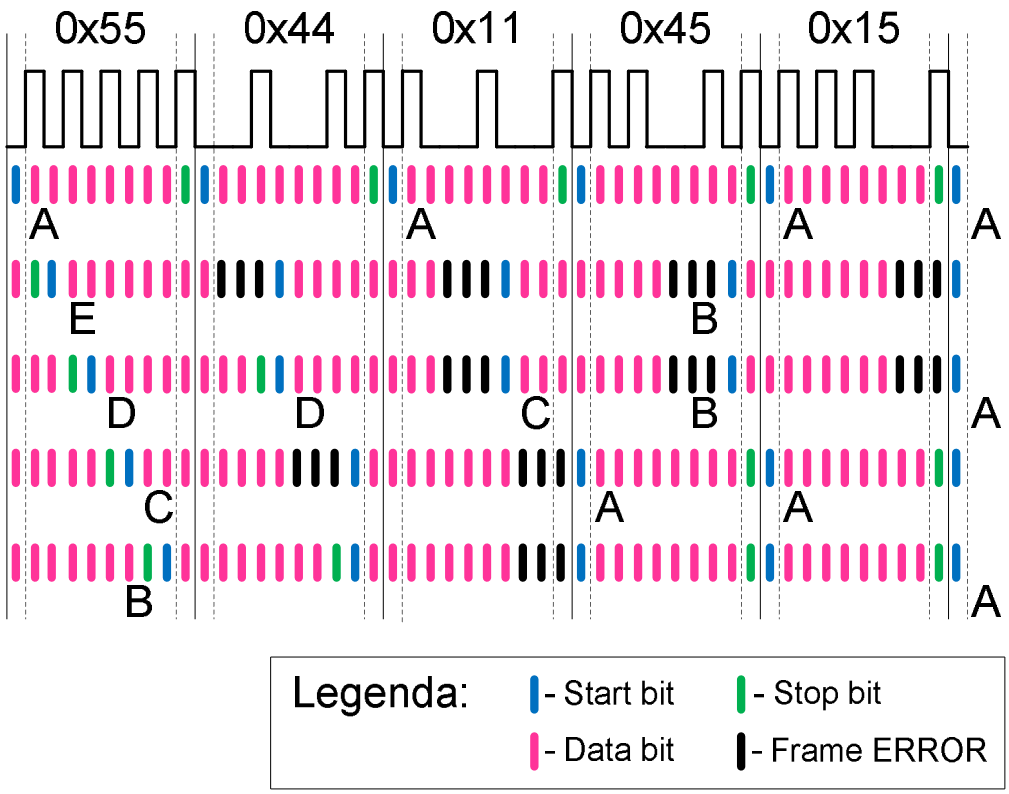


Figura 4.4 - Processo de byte-alignment

4.3.3 - *Acknowledges*

Para garantir o bom funcionamento do sistema, é necessário que o emissor obtenha uma resposta por parte do receptor a indicar se recebeu ou não o último pacote enviado, e em caso afirmativo, se recebeu correctamente ou com erros.

Dessa forma, foi implementado um sistema de *acknowledges* em que o receptor ao receber um pacote, confirma o seu *checksum* e envia de volta para o transmissor um pequeno pacote com a indicação, no campo do comando, se o pacote foi bem recebido (*checksum* calculado é igual ao *checksum* recebido no final do pacote) ou se foi recebido com erros (*checksum* calculado difere do recebido no final do pacote). Assim, é necessário que imediatamente após o envio de um pacote, o emissor passe a receptor, e imediatamente após a recepção de um pacote, o receptor passe a emissor. O emissor esperará um determinado tempo de *timeout* pela recepção do *acknowledge*, findo o qual voltará a reenviar o último pacote. Se receber um *acknowledge* a indicar a correcta recepção do pacote, o emissor enviará o pacote seguinte, mas se receber um *acknowledge* a indicar a recepção do pacote com erros, o emissor reenviará novamente o último pacote.

Na *Figura 4.5* podemos ver o *data frame* do pacote de *acknowledge*. Este pacote é apenas constituído pelo *Header*, pelo *Command* e pelo *Length* e são o exemplo de pacotes sem dados alguns no respectivo campo. O campo *Command* toma o valor 0x01 se o pacote tiver sido bem

recebido ou o valor 0x02 se este não tiver sido correctamente recebido. Não é enviado também qualquer tipo de *cheksum*.

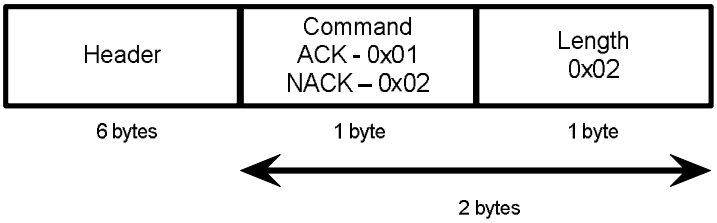


Figura 4.5 - Data frame do pacote de acknowledge

4.3.4 - Pacotes

O único pacote existente é um pacote de envio de dados simples que serve para testar os *acknowldeges*. A buzina ao receber estes dados apenas tem que enviar um *acknowledge* a sinalizar a correcta ou incorrecta recepção do pacote. Na *Figura 4.6* podemos ver o *data frame* deste pacote de testes e respectivo comando.

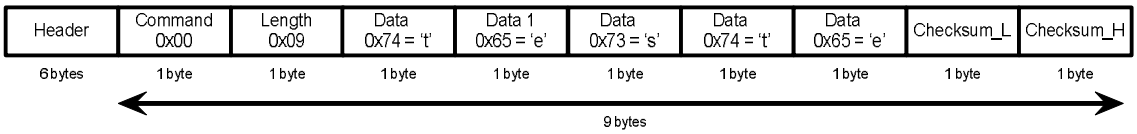


Figura 4.6 - Data frame do pacote de testes

4.3.5 - Cálculo do Checksum

O cálculo do checksum é calculado através das seguintes fórmulas:

$$CHKSUM_L = INV[0x80\text{^}('a')\text{^}('e')\text{^}Command\text{^}Data_0\text{^}Data_2\text{^}(\dots)\text{^}Data_n]$$
$$CHKSUM_H = INV[('p')\text{^}('c')\text{^}('r)\text{^}Length\text{^}Data_1\text{^}Data_3\text{^}(\dots)\text{^}Data_{n-1}]$$

, em que *'^'* significa *XOR (eXclusive OR)* e para um exemplo em que *n* é par.

Estas fórmulas são em tudo idênticas às fórmulas utilizadas para o cálculo do checksum pelo *BSL* e que foram apresentadas no primeiro capítulo.

4.3.6 - Alterações efectuadas

Mais uma vez o código foi todo revisto, analisado, corrigido e comentado a fim de o tornar legível para uma futura consulta e alteração.

Apesar dos módulos *wireless* estarem já a funcionar, era indispensável que se pudesse observar os dados recebidos via *wireless*. Assim o código foi ligeiramente alterado de modo a imprimir no *display* da buzina os *bytes* recebidos. Apesar de estes estarem a ser impressos correctamente, o simulador de *PDA* continuava a indicar que estes tinham sido recebidos, mas incorrectamente. Como os dados estavam a ser bem recebidos, o problema só poderia estar nos restantes campos do *data frame* ou no envio dos *acknowledges*. Para verificar se o problema estava nos restantes campos do *data frame*, alterou-se novamente o código de modo a que fossem agora impressos no *display* todos os seus campos. Foi então detectado e corrigido um pequeno erro no código do cálculo do *checksum* que fazia com que o *checksum* nunca fosse igual, sendo que não havia nenhum problema com o envio do *acknowledge*.

Com o primeiro teste efectuado com sucesso, faltava agora tentar enviar um pacote que fosse interpretado pela buzina como sendo um comando a enviar para a pista de luzes. Foi assim criado mais um tipo de pacote como mostra a *Figura 4.7*.

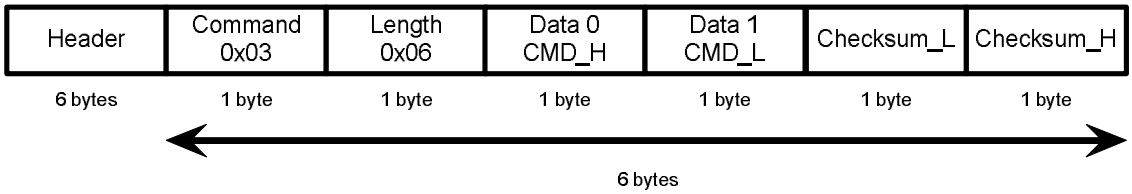


Figura 4.7 - Data frame do pacote de envio de comando para as luzes

O código da buzina teve novamente que ser ligeiramente alterado de modo a reconhecer este novo tipo de pacotes. O campo *CMD*_H corresponde ao *byte* mais significativo do comando e o campo *CMD*_L corresponde ao *byte* menos significativo do comando. Assim que a buzina reconhece o comando e verifica a validade do *checksum*, ela envia-o para a pista de luzes.

Foi possível verificar o correcto funcionamento deste comando, sendo agora possível enviar comandos cada luz individualmente, ou para todas em paralelo, através do envio destes via *wireless* pelo simulador da *PDA* e pela respectiva recepção, interpretação, validação e retransmissão por cabos da buzina para a pista de luzes.

De seguida foi criado um novo comando que escrevesse no *display LCD* com a ajuda da função previamente criada na buzina para esse efeito. O pacote criado tem o seguinte *data frame* mostrado na *Figura 4.8*. De salientar que o *Length* do *data frame* depende sempre do número de caracteres a enviar, mas este tamanho não está limitado ao número de colunas do *display LCD.* Se a posição de determinado carácter ultrapassar o limite do *display*, ele continuará a ser escrito no início da linha seguinte. Isto quer dizer que com o envio de um único pacote é possível escrever vários caracteres em várias linhas, desde que sejam consecutivas. Toda a informação existente nessas linhas será sobreposta pelos novos caracteres.

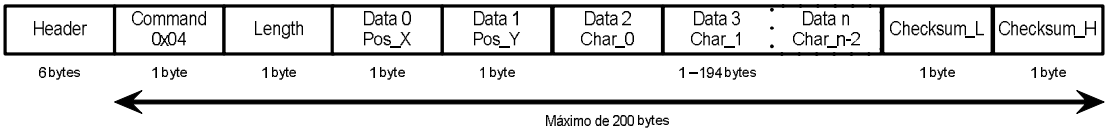


Figura 4.8 - Data frame do pacote que envia informação para escrever no display LCD

Finalmente criou-se um último pacote que serve para ligar/desligar a sirene e para definir o seu tempo total de funcionamento. O pacote tem o seguinte data frame apresentado na *Figura 4.9*. Os dois primeiros *bytes* do campo de dados representam o tempo total de funcionamento da sirene, os dois *bytes* seguintes representam o tempo em que esta estará ligada e os dois *bytes* seguintes o tempo em que estará desligada. A sirene alterna entre o estado ligado ou desligado até esgotar o tempo total definido para o seu funcionamento. Todos os tempos estão representados em milissegundos, quer isto dizer que o tempo total, o de ligado e o de desligado estão restringidos a um máximo de 65535*ms*.

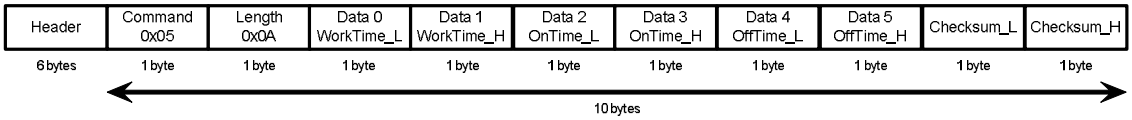


Figura 4.9 - Data frame do pacote que activa a sirene

De seguida foi completamente revisto o *byte-alignment* e verificamos que os *bytes* enviados para garantir o alinhamento não funcionavam quando se utilizavam as interrupções para receber os dados, pois a quarta das condições necessárias para garantir o *byte-alignment* segundo o algoritmo dos módulos *UART* não estava a ser cumprida. O problema que aqui se coloca é que o módulo *UART* quando detecta um *framing error* (*stop bit* a *low*, em vez de *high*) espera até receber um *bit* a *high* (ou mais *bits*, desde que consecutivos) para assumir que o seguinte *bit* a *low* é o novo *start bit* (na *Figura 4.4* isso é perfeitamente visível), enquanto no algoritmo das interrupções (ver *Figura 4.2*) ele assume imediatamente o próximo *bit* a *low* como sendo o *low start bit* (não espera assim por receber primeiro pelo menos um *high stop bit*). Vejamos a *Figura 4.10* que ilustra a ineficácia de sincronização do algoritmo das interrupções através da recepção dos 5*bytes* (0*x*55, 0*x*44, 0*x*11, 0*x*45, 0*x*15) anteriormente descritos. O algoritmo das interrupções simplesmente ignora o *byte* recebido, não conseguindo efectuar de seguida uma correcta sincronização.

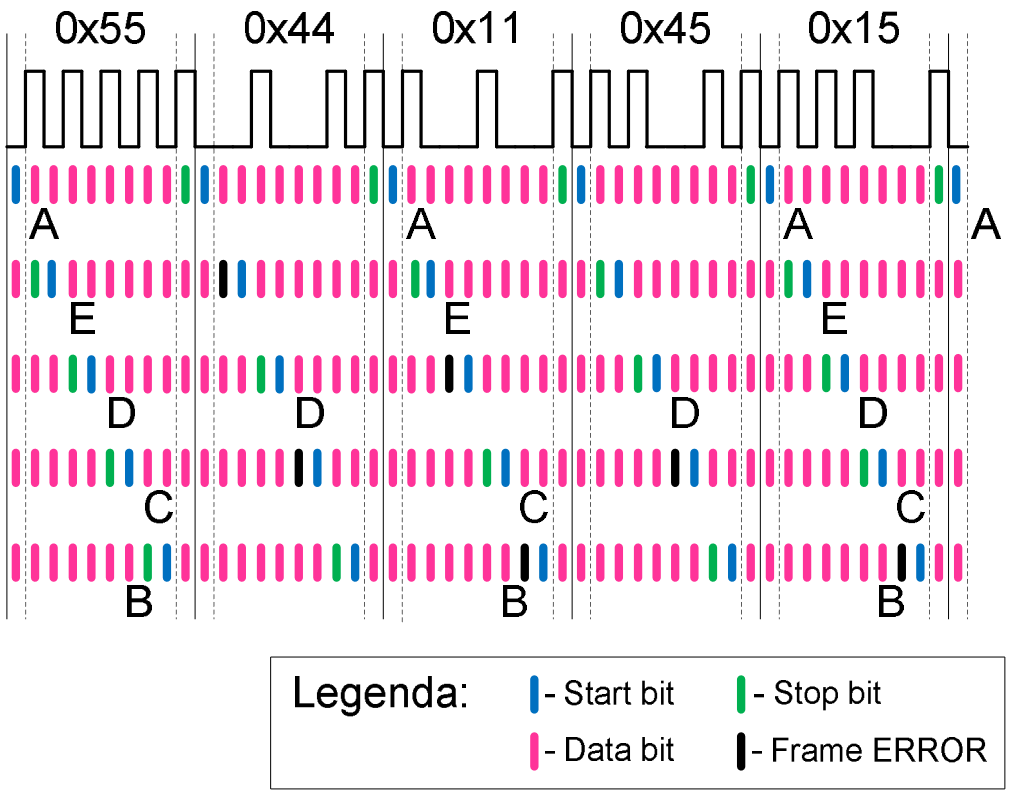


Figura 4.10 - Inexistência de byte-alignment segundo o algoritmo das interrupções

Para resolver este problema existiam duas hipóteses: ou se alterava o algoritmo das interrupções de modo a que este esperasse por pelo menos um *high stop bit* para considerar o próximo *low bit* como sendo o novo *low start bit* e mantinha-se o *byte-alignment* tal e qual como estava; ou tentava-se arranjar um novo *byte-alignment* mais simples que o anterior e que funcionasse simultaneamente para o algoritmo da *UART* e das interrupções.

No algoritmo das interrupções, se o *high stop bit* estiver incorrecto, ele procurará sempre pelo primeiro *low bit* e assume-o como sendo o próximo *low start bit*, enquanto no algoritmo da *UART*, se o *high stop bit* estiver incorrecto, ele procurará por um *high bit* e assumirá então o primeiro *low bit* que aparecer como sendo o novo *low start bit*.

Se ignorarmos a primeira das quatro condições do algoritmo da *UART* (que garante apenas que não se verifique uma possível situação de incorrecto balanceamento no módulo *RF*) chegamos a um modo de sincronização muito fácil: basta enviar um *byte* com o valor 0xFF antes do envio do *Header* de cada pacote para garantir o *byte alignment*. Na *Figura 4.11*, podemos visualizar o funcionamento detalhado deste novo modo, tanto para o algoritmo da *UART* como para o algoritmo das interrupções. Para todas as 10 situações possíveis, verificamos que o alinhamento é executado correctamente.

Em resumo, com este novo método de *byte-alignment* através do envio da palavra 0xFF, acontece uma de três situações:

- *Byte-alignment* vem correcto e continua correcto (caso A);
- *Byte-alignment* vem desfasado em um *bit*, nenhuma palavra é validada e o *byte-alignment* é garantido pelo *low start bit* da próxima palavra (caso B);
- *Byte-alignment* vem desfasado em mais de um *bit*, uma palavra aleatória será validada e o *byte-alignment* é garantido pelo *low start bit* da próxima palavra (todos os casos restantes).

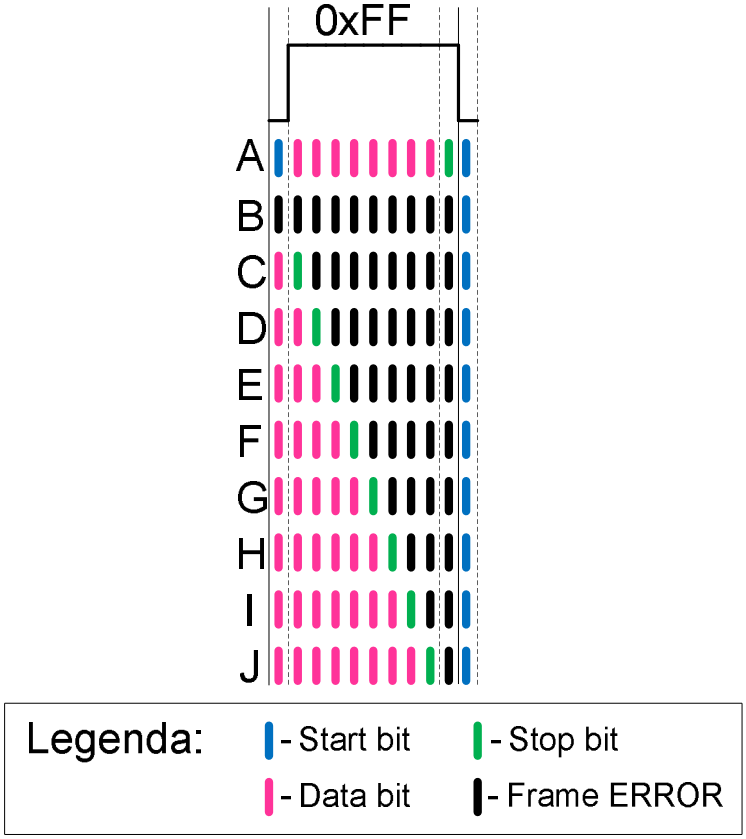


Figura 4.11 - Solução do byte-alignment para o algoritmo da UART e para o das interrupções

Com esta nova solução, não foi necessário adicionar mais código ao algoritmo das interrupções, e reduzimos em quatro o número de *bytes* a enviar para garantir a sincronização.

Tanto com o código original, como com este novo código, detectamos que por vezes a buzina, quando configurada em modo de recepção, deixava de conseguir detectar os dados enviados pelo simulador de *PDA*. Analisando o código em modo de *debugging* através do *IAR Embedded Workbench*, verificamos que a buzina deixava de detectar os *bytes* recebidos por *RF*. Para evitar esta situação, decidimos criar uma rotina de controlo no receptor que faz o *reset* do seu módulo *RF* sempre que este fique um determinado tempo sem receber dados para garantir que esses não estão a ser recebidos devido ao facto de este ter deixado, por alguma razão, de os reconhecer. Esta solução, apesar de não poder ser definitiva, solucionou este problema.

4.4 - Conexão *RS232*

A *PDA* comunicará com o módulo wireless através do microcontrolador existente no simulador de *PDA*. A comunicação entre o módulo *wireless* e o microcontrolador já foi aqui apresentado e iremos agora discutir a ligação *RS232* entre a *PDA*, o *MAX232* e o microcontrolador. Na *Figura 4.12* podemos observar as ligações existentes entre o microcontrolador, o módulo *RS232* e a *PDA*.

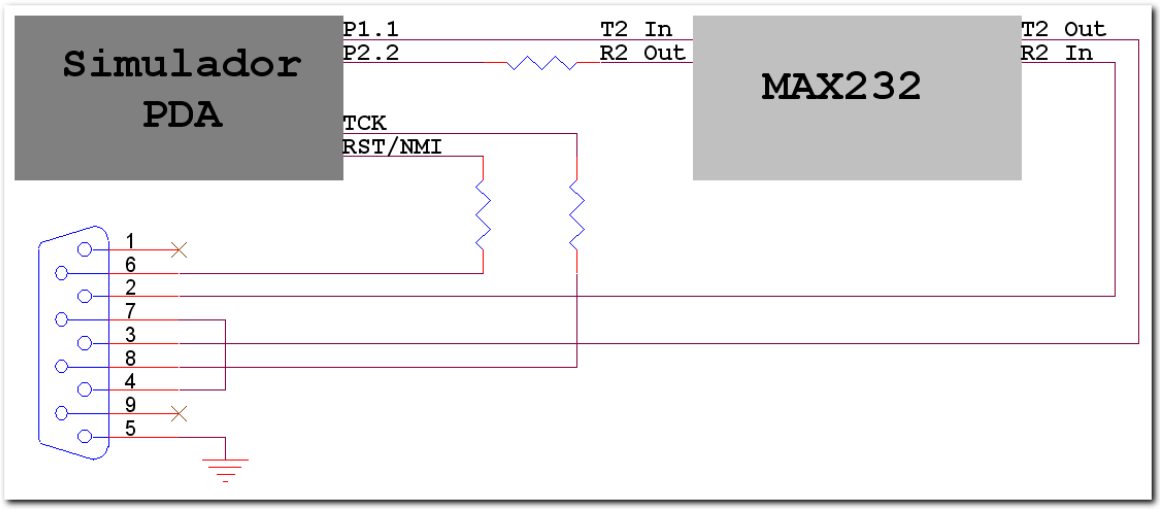


Figura 4.12 - Diagrama de ligações entre o microcontrolador, o MAX232 e a PDA através de porta série

4.5 - *Firmware Updates*

Também este módulo terá a possibilidade de receber *firmware updates*. Estes *firmware updates* serão efectuados através da ligação existente por *RS232* entre a *PDA* e o microcontrolador. Os sinais de controlo dos pinos *TCK* e *RST/NMI* serão enviados pela *PDA* através do conector de porta série, e a informação será recebida pelo microcontrolador através do pino *P2.2* e os *acknowledges* enviados através do pino *P1.1*, tal como se pode verificar pela *Figura 4.12*.

Capítulo 5. Reprogramador da Buzina

5.1 – Introdução

Neste capítulo iremos demonstrar detalhadamente o funcionamento do reprogramador da buzina. Será apresentado todo o trabalho realizado anteriormente e todas as alterações efectuadas durante a realização deste projecto. De realçar que o *software* do reprogramador da buzina disponibilizado no início do projecto também não se encontrava a funcionar.

Para permitir que *firmware updates* fossem efectuados com facilidade no microcontrolador da buzina, foi necessário adicionar um microcontrolador adicional no módulo da buzina que tivesse como única finalidade a recepção por *wireless* do novo *firmware*, e o enviasse através de *BSL*. Para tal, será necessário que este novo microcontrolador consiga inicializar e configurar o módulo *wireless*, pois este não é capaz (ao contrário de outros módulos como o *XBee*, por exemplo) de se iniciar e configurar sozinho, a fim de receber o novo *firmware* e configure e inicialize o *BSL* no microcontrolador da buzina.

A necessidade de adicionar este novo microcontrolador surge da obrigatoriedade de interpretar a informação recebida pelo módulo *wireless* a fim de filtrar os dados pertencentes ao novo *firmware*, dos dados que contenham comandos para os restantes módulos. Esta interpretação terá ser efectuada por um microcontrolador e nunca poderá ser efectuada pelo próprio

microcontrolador da buzina pois este terá que se encontrar em modo *BSL* para que seja reprogramado com o novo *firmware*. Em princípio o programa deste novo microcontrolador será tão básico que não será necessário que sejam efectuados *firmware updates* a ele próprio, embora fosse possível estabelecer ligações entre a buzina e o reprogramador de forma que esta, ao receber pacotes com um novo *firmware* para o reprogramador, activasse a sequência de entrada em *BSL* neste último.

5.2 - Buzina

A comunicação com a buzina é efectuada através de 6 pinos como se pode ver pela *Figura 5.1*. Dois desses pinos servem para fornecer a alimentação ao reprogramador da buzina e os outros quatro servem para a estabelecer a comunicação entre ambos os módulos.

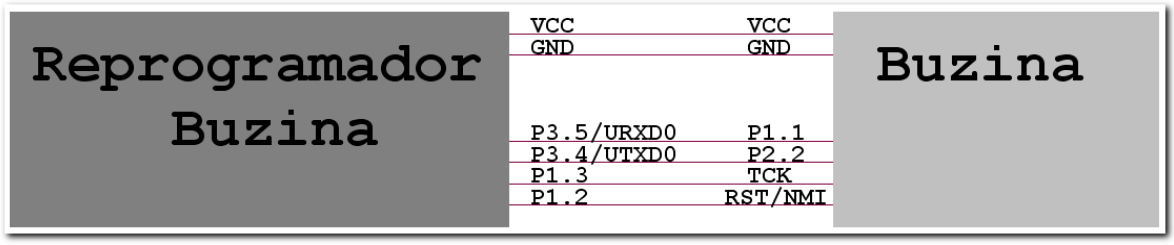


Figura 5.1 - Esquema de ligações entre o reprogramador da buzina e a buzina

A interacção entre o reprogramador da buzina e a própria buzina ainda não tinha sido implementado, apenas o *hardware* estava pronto e correctamente conectado.

5.2.1 - Alterações efectuadas

Foi necessário alterar o código da buzina e do seu reprogramador para evitar que o funcionamento de um impeça o funcionamento em simultâneo do outro. Para tal, definimos que quando se liga o sistema, o reprogramador inicia em modo *Master* e a buzina em modo de *SIave*.

Quer isto dizer que o reprogramador é o responsável por configurar e iniciar o módulo *wireless*, ficando à espera do primeiro pacote de dados. Se o primeiro pacote não for um pacote de sinalização do início do envio de novo *firmware* para a buzina, o reprogramador entra imediatamente em *LPM* e a buzina assume o controlo e configuração do módulo *wireless*. Se esse pacote for um pacote de sinalização do início do envio de novo *firmware* para a buzina, o reprogramador inicializa o *BSL* na mesma e envia o novo *firmware update*. No final entra em *LPM* e entrega o controlo e configuração do módulo *wireless* à buzina.

Foi utilizado o pino *P3.5* do reprogramador, conectado ao pino *P1.1* da buzina, para permitir esta definição de *Master* e *Slave*. Este deverá estar sempre a *input* na buzina para que o reprogramador sinalize, colocando-o a *high*, o seu estado de *Master* ou o seu estado de *LPM* (*Slave*), colocando-o a *low*. A buzina lê então esse pino e assumirá o papel de *Slave* se o pino estiver a *high* ou o papel de *Master* se o pino estiver a *low*.

A entrada em *BSL* é forçada pelo reprogramador da buzina assim que recebe um pacote via *wireless* a indicar o início da transmissão de novo *firmware* para a buzina. Os pinos de entrada em *BSL* são o pino *P1.2* que está conectado ao pino *RST/NMI* e o pino *P1.3* que está conectado ao pino *TCK*. Para enviar os dados é utilizado o pinos *UTXD0* e para receber os eventuais *acknowledges* do *BSL* o pino *URXD0*.

Para já, um *firmware update* do *software* da buzina apenas pode ser efectuado mal se liga o sistema, pois o primeiro pacote a receber por parte do reprogramador da buzina terá que ser um pacote que sinalize o início do envio de novo *firmware* para a buzina, caso contrário o reprogramador entra em *LPM* e a buzina continuará sempre activa até que se reinicie o sistema.

5.3 - Módulo *Wireless*

Será através do módulo *wireless* que o reprogramador da buzina receberá o novo *firmware* da buzina. Sempre que é detectado o envio de um novo *firmware*, o microcontrolador da buzina entra em *LPM* enquanto o microcontrolador do reprogramador da buzina assumirá o controlo e configurará o *CC1000* para que este funcione como receptor com os parâmetros adequados de *baud rate*, frequência de recepção, etc.

A ligação entre o microcontrolador e o *CC1000* faz-se normalmente com o auxílio de 7 pinos de *I/O*. Mais uma vez, o envio e a recepção de informação entre os vários constituintes do reprogramador da buzina utilizará como protocolo de comunicação as interrupções com regras do *RS232*, onde enviamos um *low start bit* e um *high stop bit* a delimitar cada *byte* de dados.

Neste caso, será utilizado o sinal de *clock* disponível no pino *DCLK* do módulo *wireless* para originar uma interrupção com o objectivo de efectuar a recepção ou envio de um *bit* de cada vez. O pino *DIO* do módulo *wireless* estará ligado ao pino *P2.4* do reprogramador da buzina e este estará configurado como *input* (e o pino *DIO* como *output*) se estivermos a receber dados, ou como *output* (e o pino *DIO* como *input*) se estivermos a transmitir dados. A *Figura 5.2* mostra-nos a interface de ligação entre o módulo *wireless* e o microcontrolador.

A *Figura 5.3* mostra-nos o diagrama da rotina de atendimento à interrupção que efectua a recepção ou transmissão de um *bit* de dados. Os *buffers RxBuffer* e *TxBuffer* são *buffers* onde está contido o *byte* acabado de receber ou a enviar, respectivamente.

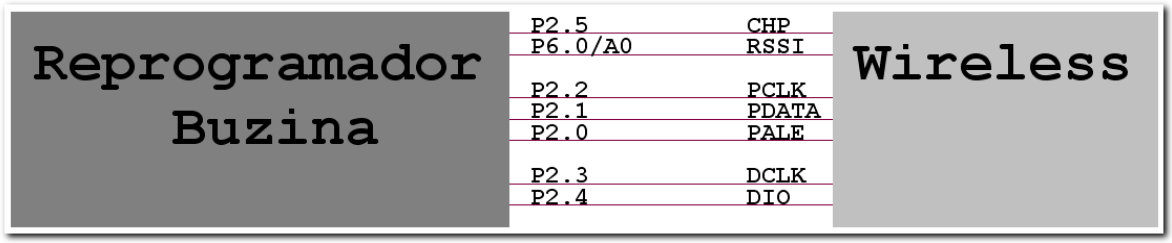


Figura 5.2 - Interface de ligação entre o microcontrolador e o módulo wireless do simulador de PDA

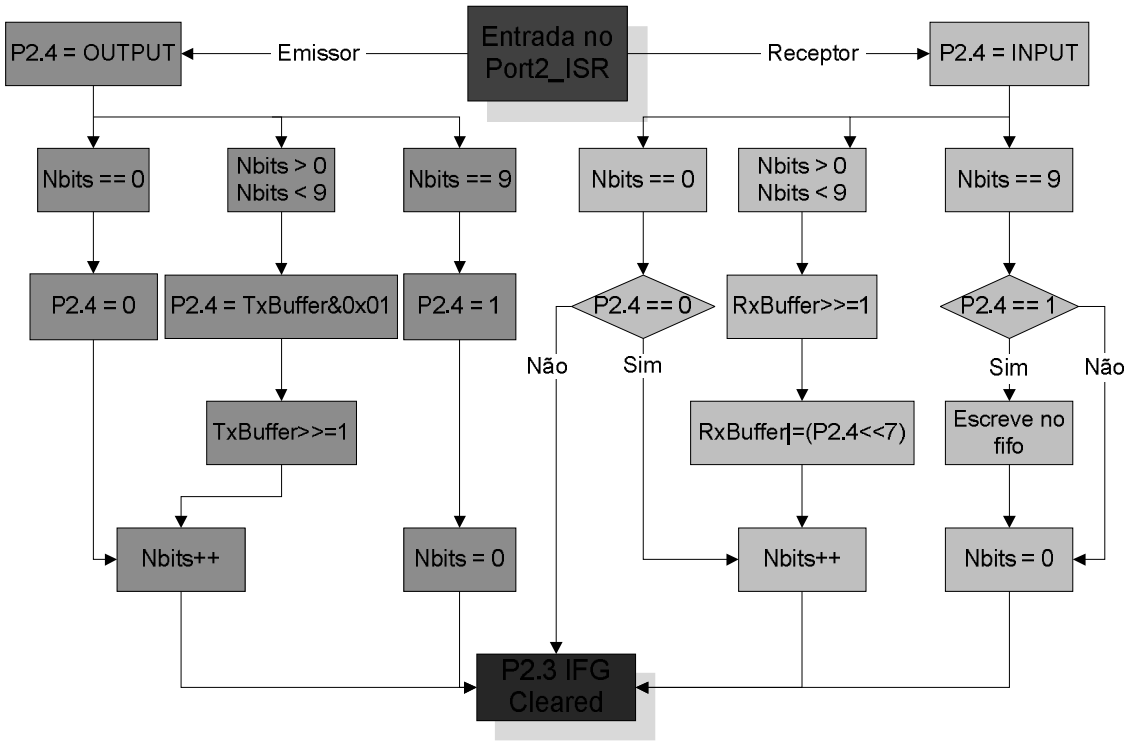


Figura 5.3 – Diagrama da rotina de atendimento à interrupção no pino P2.3

De salientar ainda que, como explicado anteriormente, é necessário recorrer a um *buffer* adicional do tipo *fifo* para armazenar os *bytes* recebidos.

Para comprovar se a comunicação *wireless* está bem definida e os pacotes estão a ser recebidos com sucesso, existem dois *LED* ligados a dois pinos de *I/O* do microcontrolador que podem ser configurados para retratar o estado do *acknowldege* a ser enviado para o transmissor após a recepção dos pacotes. Na *Figura 5.4* podemos ver o esquema de ligação dos dois *LED* mencionados.

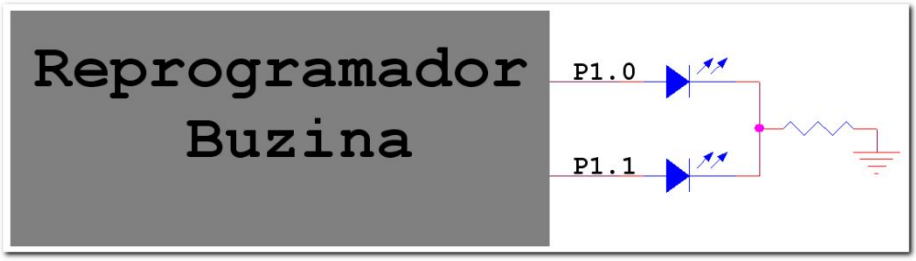


Figura 5.4 - Ligação dos LED

5.3.1 - Alterações efectuadas

Corrigindo alguns erros na configuração do módulo *wireless* e nas funções responsáveis pela recepção e pelo tratamento de dados por parte do reprogramador, foi possível detectar, com a ajuda do osciloscópio, a correcta recepção dos pacotes de dados. Tivemos neste caso que recorrer ao osciloscópio, pois ao contrário da buzina, o reprogramador não tem acesso ao *display LCD* onde poderia facilmente imprimir a informação recebida. Finalmente, com a ajuda dos dois *LED* acima mencionados, conseguimos verificar o estado do *acknowledge* que o reprogramador enviava de volta para o simulador da *PDA* (primeiro *LED* acesso significava pacote recebido sem erros, o segundo *LED* acesso significava pacote recebido mas com erros, e nenhum *LED* acesso significava nenhum pacote recebido).

5.4 - Pacotes *wireless*

Com a comunicação por *wireless* a funcionar correctamente, podemos finalmente tentar enviar os pacotes com o novo *firmware* da buzina. Assim como as funções necessárias para que houvesse interacção entre o reprogramador da buzina e a buzina, também estes pacotes ainda não estavam criados.

5.4.1 - Alterações efectuadas

Para poder receber os pacotes com o novo *firmware*, foi necessário criar também no reprogramador da buzina um vector capaz de armazenar até um máximo de 200*bytes* (200*bytes* é o tamanho máximo definido para um pacote de dados e será também utilizado como tamanho máximo para os pacotes contendo o novo *firmware*). Desta forma, a estrutura do *data frame* dos pacotes de novo *firmware* será igual à estrutura do *data frame* dos pacotes de dados e é apresentada na *Figura 5.5*. Foi assim possível utilizar no reprogramador da buzina as mesmas funções, com pequenas alterações, utilizadas previamente na buzina para detectar e receber os pacotes de dados.

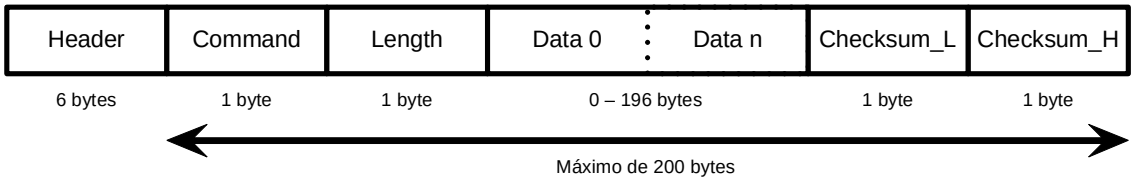


Figura 5.5 - Data Frame dos pacotes de novo firmware

Foi depois necessário definir o processo de sinalização de início do envio dos pacotes contendo o novo *firmware* e o processo de sinalização do fim de envio desses pacotes. Como é o campo *Command* que distingue os diferentes tipos de pacotes, concluiu-se que não existia necessidade de definir um novo pacote de sinalização de inicio do envio de novo *firmware*, pois cada pacote contendo o novo *firmware* terá um campo *Command* igual entre si, mas distinto de todos os outros pacotes criados até agora. Bastava então criar um pacote que sinalizasse o fim do envio do novo *firmware*. Na *Figura 5.6* podemos ver o *data frame* dos pacotes que contêm o novo *firmware*, em que o campo *Data0* e *Data1* contêm os *bits* menos e mais significativos, respectivamente, do endereço de memória onde começar a escrever os dados contidos nos restantes campos de dados. Na *Figura 5.7* observamos o *data frame* do pacote que sinaliza o fim do envio de novo *firmware*.

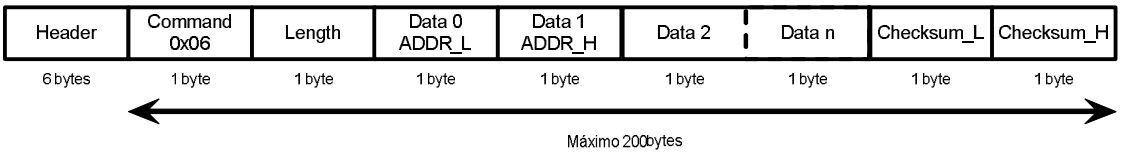


Figura 5.6 - Data frame dos pacotes contendo o novo firmware da buzina

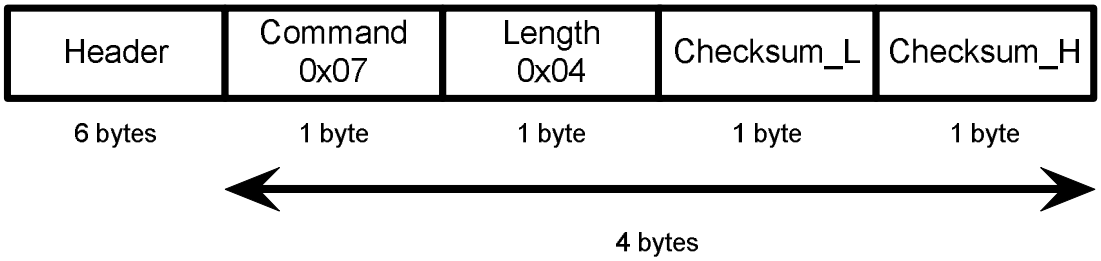


Figura 5.7 - Data frame do pacote que sinaliza o fim do envio do novo firmware da buzina

Se o reprogramador receber então como primeiro pacote um pacote que contenha no campo *Command* o valor 0x06, ele saberá que vem aí um *firmware update* para a buzina e trata de iniciar a sequência de entrada em *BSL*. Por cada pacote recebido, o reprogramador envia de seguida os dados por *BSL* para a buzina e só depois fica pronto para receber um novo pacote.

Com a recepção do pacote que sinaliza o fim de envio do *firmware update*, o reprogramador efectua o *reset* à buzina, a fim de ela reiniciar já com o novo *firmware* a correr, e entra em *LPM*.

O sistema foi testado e comprovou-se o seu bom funcionamento. Procedeu-se então ao comentário de todo o código para facilitar uma futura análise e alteração do código existente.

Capítulo 6. Apresentação e Discussão de Resultados

6.1 - Introdução

Neste capítulo iremos apresentar todos os cálculos efectuados e resultados obtidos. Iremos abordar os cálculos dos valores das resistências utilizadas, o tamanho (em *bytes*) do programa de cada microcontrolador e suas principais implicações, o alcance máximo determinado para os módulos *wireless*, o tempo máximo necessário para transmitir um pacote de 200*bytes* de dados e receber o seu respectivo *acknowledge* e a redução no consumo de potência quando se utiliza o *power-management* na buzina.

6.2 - Cálculo do valor das resistências da pista de luzes

A resistência utilizada para limitar a corrente ao longo da pista de luzes aquando do seu endereçamento foi determinada experimentalmente. Para uma pista composta por 26 luzes, observou-se que era necessário uma resistência de 67Ω e a corrente que a percorre é de é de 3.3/67 = 49mA. É este valor de corrente que implica a necessidade de termos vários pinos

ligados entre si em cada pino *I/O_R* de cada luz. Um aumento do número de luzes poderá obrigar a um valor de *R* ainda mais baixo, de qualquer forma, à partida uma pista de luzes nunca terá mais de 26 luzes.

6.3 - Cálculo do valor da resistência da buzina

Já a resistência adicional que foi necessário colocar na buzina para possibilitar a detecção de avaria na primeira luz foi calculada matematicamente. Temos que garantir que quando não existe avaria na primeira luz, a queda de tensão nessa resistência terá que a tensão presente no pino *CMP* (ligado a um dos terminais dessa mesma resistência) da buzina seja inferior a $VCC/4$. Da mesma forma, quando existe avaria na primeira luz, a queda de tensão terá que ser tal que a tensão presente no pino *CMP* da buzina seja superior a $VCC/4$. Este problema é facilmente resolvido pelo seguinte sistema de equações:

$$\begin{cases} \frac{67}{67+R} \times VCC < \frac{VCC}{4} \\ \frac{67+67}{67+67+R} \times VCC > \frac{VCC}{4} \end{cases} \Leftrightarrow \begin{cases} \frac{67}{67+R} \times \cancel{VCC} < \frac{\cancel{VCC}}{4} \\ \frac{67+67}{67+67+R} \times \cancel{VCC} > \frac{\cancel{VCC}}{4} \end{cases} \Leftrightarrow \begin{cases} 67 \times 4 < 67 + R \\ 134 \times 4 > 134 + R \end{cases}$$

$$\Leftrightarrow \begin{cases} R > 201\Omega \\ R < 402\Omega \end{cases}$$

Assim a melhor opção seria uma resistência que se aproximasse do valor de 300Ω, pois é o valor intermédio entre o mínimo e máximo calculado. Como as resistências a que temos acesso de valor mais próximo do pretendido são de 270Ω ou 330Ω, e como ambas estão à mesma distância do valor de 300Ω, tivemos que fazer contas para ver aquela que melhor se adequava à nossa situação:

$$V_{270\Omega} = \frac{67}{67+270} \times VCC \cong 0.1988 \times VCC \cong 0.6560V$$

$$V_{270\Omega} = \frac{67+67}{67+67+270} \times VCC \cong 0.3317 \times VCC \cong 1.0946V$$

$$V_{330\Omega} = \frac{67}{67+330} \times VCC \cong 0.1688 \times VCC \cong 0.5570V$$

$$V_{330\Omega} = \frac{67+67}{67+67+330} \times VCC \cong 0.2888 \times VCC \cong 0.9530V$$

$$I_{270\Omega} = \frac{3.3}{270} \cong 12.22mA$$

$$I_{330\Omega} = \frac{3.3}{330} \cong 10.00mA$$

$$P_{270\Omega} = \frac{3.3^2}{270} \cong 40.33mW$$

$$P_{330\Omega} = \frac{3.3^2}{270} \cong 33.00mW$$

$$\frac{VCC}{4} \cong 0.825V$$

	270Ω	**330Ω**
ΔV (luz avariada)	$0.169V$	$0.2696V$
ΔV (luz não avariada)	$0.268V$	$0.128V$
I	$12.22mA$	$10.00mA$
P	$40.33mW$	$33.00mW$

Tabela 6.1 - Características das resistências

Analisando os valores acima referidos, verificamos que a resistência de 330*Ω* apresenta um menor valor de potência dissipada e de corrente em relação à resistência de 270*Ω*. Em contrapartida, a resistência de 270*Ω* apresenta maiores margens de tensão em relação a $VCC/4$ do que a resistência de 330*Ω*. Como a diferença de correntes e de tensão é muito baixa, e como a sensibilidade a ruído e a pequenas oscilações de tensão é muito importante aquando do endereçamento (ruído e oscilações de corrente/tensão podem elevar ou diminuir a tensão no pino *CMP* e a comparação ser mal efectuada), optou-se pela resistência de 270*Ω*, pois é aquela que apresenta maior robustez.

6.4 - Cálculo do valor das resistências para detecção de avaria em duas luzes consecutivas

Para detectar avaria em dois microcontroladores consecutivos, foi necessário alterar ligeiramente o circuito de cada luz da pista de luzes. Neste novo circuito foram adicionas duas resistências denominadas *R1* e *R2*, responsáveis por criar uma queda de tensão no pino *CA1* do comparador do microcontrolador. Com o auxílio da *Figura 2.37*, iremos calcular de seguida os valores de *R1* e de *R2* para os quais o valor presente no pino *CA1* é superior a $VCC/4$, no caso de existir apenas um microcontrolador avariado, ou inferior a $VCC/4$, no caso de existirem dois microcontroladores consecutivos avariados. Para todas as análises, partiremos do pressuposto que $(R1 + R2) >> R$, de modo a que as aproximações efectuadas sejam válidas.

Analisando o caso de existir apenas um microcontrolador avariado, chegamos à seguinte condição:

$$\frac{VCC}{2} \times \frac{R_2}{R_2 + R_1} > \frac{VCC}{4} \Leftrightarrow \frac{R_2}{R_2 + R_1} > \frac{2}{4} \Leftrightarrow 4 \times R_2 > 2 \times R_2 + 2 \times R_1 \Leftrightarrow R_2 > R_1$$

Analisando agora o caso de existirem dois microcontroladores avariados, chegamos à seguinte condição:

$$\frac{VCC}{3} \times \frac{R_2}{R_2 + R_1} < \frac{VCC}{4} \Leftrightarrow \frac{R_2}{R_2 + R_1} < \frac{3}{4} \Leftrightarrow 4 \times R_2 < 3 \times R_2 + 3 \times R_1 \Leftrightarrow R_2 < 3R_1$$

Concluindo, temos duas condições gerais para que o circuito funcione correctamente:

$$(R_1 + R_2) \gg R \quad \text{e} \quad R_1 < R_2 < 3 \times R_1 \quad \text{, com } R = 68\Omega$$

Para efectuar os testes na placa branca, seleccionamos $R_1 = 10k\Omega$ e $R_2 = 8k\Omega$.

6.5 - Tempo de envio de pacote de 200bytes e recepção do respectivo acknowledge

O tempo necessário para o envio de um pacote de 200*bytes* e a recepção do respectivo *acknownledge*, medido para a versão do *software* disponibilizada no início do projecto, é de cerca de 2000*ms*. A nova versão do *software* conseguiu reduzir este tempo para aproximadamente 750*ms*. A transmissão do pacote de 200*bytes* demora sensivelmente 217*ms* e a transmissão do *acknowledge* demora sensivelmente 6*ms*. Depois é dado ainda 500*ms* para que o módulo transmissor fique em modo receptor para receber o *acknowledge*, para que o módulo receptor fique em modo transmissor a fim de enviar o *acknowledge*, para que ambos os módulos voltem depois à sua configuração inicial e para que a buzina tenha tempo de tratar todos os dados recebidos por *wireless*. Passamos agora de seguida a explicar os cálculos efectuados.

$2bytes$ (*byte-alignment*) + $6bytes$ (*header*) + $200bytes$ (pacote) = $208bytes$

$208bytes \times 10bits$ ($1bit$ *start* + $8bits$ *data* + $1bit$ *stop*) = $2080bits$

$$\frac{2080}{9600} \cong 217ms$$

$2bytes$ (*byte-alignment*) + $6bytes$ (*header*) + $2bytes$ (pacote) = $10bytes$

$10bytes \times 10bits$ ($1bit$ *start* + $8bits$ *data* + $1bit$ *stop*) = $100bits$

Como os *acknowledges* são pacotes muito pequenos e como foi detectada experimentalmente alguma dificuldade por parte do simulador da *PDA* em receber esses *acknowledges* correctamente, foi decidido que em vez de 1, seriam enviados 5 *acknowledges* para diminuir a probabilidade de o pacote ser bem recebido mas o seu *acknowledge* não, assim:

$$100bits \times 5 = 500bits$$

$$\frac{500}{9600} \cong 6ms$$

O tempo de 500*ms* para a buzina tratar os dados recebidos, foi também ele obtido experimentalmente.

6.6 - Alcance máximo

Para determinar o alcance máximo de comunicação entre o módulo *wireless* da buzina e o módulo *wireless* do simulador da *PDA*, foram testados dois módulos *MMcc1000*, um deles configurado para funcionar a 433*MHz* e um outro para funcionar a 868*MHz*.

Colocou-se então a buzina (ligada à pista de luzes) junto da janela de uma sala no 2.º piso do departamento e fomos para o exterior com o simulador da *PDA*. As janelas da sala encontravam-se fechadas e a buzina foi colocada a aproximadamente 1*m* desta, à altura do parapeito. A buzina encontrava-se assim a, aproximadamente, 5*m* de altura em relação ao simulador da *PDA*.

Notamos que à medida que íamos aumentando a distância entre ambos os módulos, a taxa de sucesso de envio dos pacotes ia diminuindo. Então decidimos utilizar, como critério para determinar a distância máxima, um valor mínimo de 33% para a taxa de comunicação. Quer isto dizer que íamos afastando o simulador de *PDA* da buzina enquanto a taxa fosse superior a 33% e anotávamos alguns pontos de referência visuais a que tal ocorria.

Depois recorremos ao *Google Earth*, identificávamos esses pontos de referência visuais e procedemos à medição da distância com a ajuda da ferramenta *Ruler* existente no programa.

Na *Figura 6.1* podemos ver a medição do alcance máximo medido para o módulo de 433*MHz* e na *Figura 6.2* a medição do alcance máximo medido para o módulo de 868*MHz*.

Figura 6.1 – Distância máxima medida para o módulo de 433MHz

Figura 6.2 - Distância máxima medida para o módulo de 868MHz

Tendo determinado uma aproximação da distância máxima de comunicação entre os dois módulos, decidimos criar uma tabela de referência com o alcance máximo para todos os valores de potência. Para tal, programamos o simulador da *PDA* com os vários valores de potência possíveis, deixando a buzina a transmitir os *acknowledges* em potência máxima, e medimos, segundo o mesmo critério utilizado anteriormente, o alcance máximo para cada uma. Estes testes foram efectuados para ambos os módulos de 433*MHz* e de 868*MHz* numa zona completamente horizontal e sem qualquer tipo de obstáculo entre os dois módulos ou perto destes. Na *Tabela 6.2* e no *Gráfico 6.1* estão representados os valores medidos para o alcance máximo de cada potência de transmissão (a potência máxima de transmissão no módulo de 868*MHz* está limitada a 5*dBm*).

	433MHz	**868MHz**
Potência [dBm]	Alcance máximo [m]	
-20	3	3
-19	3	3
-18	13	8
-17	13	8
-16	13	10
-15	18	15
-14	20	18
-13	25	20
-12	30	23
-11	40	25
-10	45	25
-9	55	35
-8	65	40
-7	65	50
-6	70	55
-5	75	65
-4	85	70
-3	90	70
-2	95	80
-1	100	85
0	110	95
1	150	95
2	180	100
3	220	110
4	270	115
5	290	120
6	300	-
7	300	-
8	310	-
9	320	-
10	320	-

Tabela 6.2 - Alcance máximo medido para as várias potências de transmissão

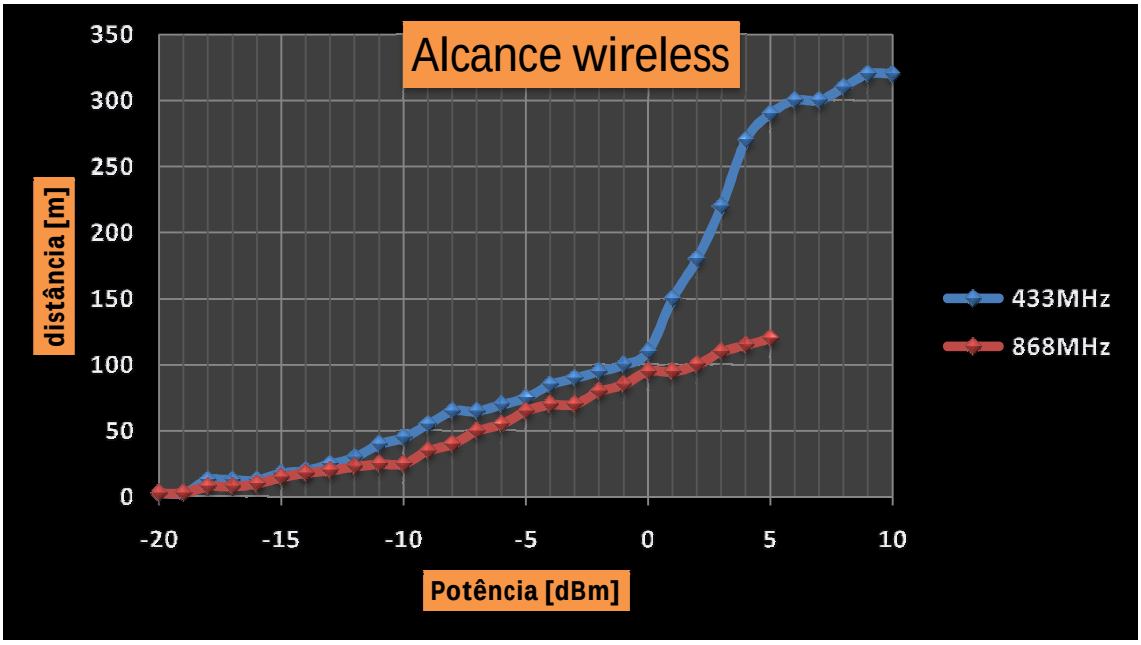


Gráfico 6.1 - Alcance máximo medido para as várias potências de transmissão

Pela análise destes dados, podemos verificar que o módulo de 433*MHz* apresenta um alcance máximo superior ao módulo de 868*MHz*. Esta diferença é mais evidente quando aumentamos a potência de transmissão acima dos 0*dBm*.

De referir também que durante os testes, verificamos que o alcance atinge o seu máximo quando nos viramos de frente para o receptor (neste caso a buzina) e seguramos o transmissor (simulador de *PDA*) na horizontal (se ficarmos virados de costas para o receptor, a taxa de sucesso de transmissão é menor). Verificamos também que se existirem grandes obstáculos perto do receptor (tais como paredes, ou objectos de grandes dimensões) ou entre ambos, essa mesma taxa de sucesso de transmissão de pacotes diminui significativamente. Verificamos também que atingimos um alcance superior quando o receptor estava colocado acima do emissor (quando este estava no departamento, o alcance máximo foi de 350*m*, enquanto que nos testes com ambos à mesma alturao alcance máximo desceu para 320*m*). Em conclusão, podemos sugerir que a buzina seja colocada no centro do topo da piscina, equidistante das paredes laterais, preferencialmente alguns metros acima do nível do solo, e que aquando da transmissão de dados por parte do treinador, este esteja virado de frente para a buzina, de modo a aumentar a taxa de sucesso de transmissão dos pacotes de dados.

6.7 - Consumo de corrente/potência

Para aumentar ainda mais a autonomia do circuito, decidimos implementar uma rotina na buzina que coloca o *transceiver* em *LPM* sempre que este não receba pelo menos um pacote num espaço de tempo de 12*s*. Depois de entrar em *LPM*, o *transceiver* é acordado a cada 6*s* e espera por um novo pacote durante 3*s*. Se não receber nenhum pacote, volta a entrar em *LPM* por mais 6*s*, mas

se receber pelo menos um pacote, fica à espera por um novo durante 12*s*. Estes tempos podem ser facilmente alterados mais tarde, o importante aqui é confirmar que o algoritmo esteja a funcionar correctamente.

6.8 - Dimensão do código de cada módulo

No final, com todo o sistema a funcionar e a comunicar entre si (excepto com a *PDA*), o tamanho total do código de cada módulo é o seguinte:

- Luzes – 930*bytes*, com código optimizado manualmente, mas sem utilizar as ferramentas de optimização do *IAR* para manter uma boa legibilidade nos *debuggings*.
- Buzina – aproximadamente 6600*bytes*, com código não optimizado (de salientar que o código da buzina já contem o próprio código das luzes)
- Reprogramador da buzina – aproximadamente 3500*bytes*, com código não optimizado
- Simulador da *PDA* – O simulador da *PDA* ainda não se encontra na sua versão final, pois a *PDA* ainda não está conectada a ele, sendo assim, não faz sentido expor para já o tamanho do código deste módulo pois ele encontra-se muito maior e muito longe ainda da sua versão final)

6.9 - Desvio do *RTC*

Para medir o desvio aproximado do *RTC*, comparamos o tempo apresentado pela buzina ao fim de 24*h* de execução contínua com o tempo apresentado com o relógio digital apresentado pelo *Microsoft Windows XP* e com um relógio de pulso. É verdade que não conhecemos à partida o erro de ambos, mas partimos do princípio que o erro apresentado pelo relógio de pulso e pelo relógio digital do *Microsoft Windows XP* será bastante baixo. De futuro, será necessário realizar testes mais precisos para garantir um erro máximo de 100μs por hora no *RTC* da buzina, mas com este teste apenas pretendíamos ter uma ideia de qual seria o seu desvio actual.

Ao fim de 24*h* de utilização contínua, não conseguimos medir qualquer desfasamento entre o relógio de pulso e o relógio do *Microsoft Windows XP*. Já o *RTC* apresentou um desvio positivo de cerca de 48*s* em relação a ambos, o que indica que o *RTC* apresenta actualmente um erro de +2*s* por hora.

Este erro pode ser facilmente corrigido, aumentando o valor do contador no *timer* do microcontrolador (definido anteriormente com o valor 4000) ou fazendo uma função, no *software* da buzina, que corrija o valor do *ctime* em intervalos de tempo bem definidos.

Como exemplo de correcção, aumentamos o valor do contador para 4002 e observamos que o desvio positivo verificado ao fim de 24*h* de utilização diminuiu para +5*s* aproximadamente, o que corresponde a um erro aproximado de +200*ms* por hora.

Capítulo 7. Conclusões e Trabalho Futuro

7.1 - Conclusões

Foi necessário empregar muito tempo para compreender e corrigir o código disponibilizado no início do projecto. Este não estava devidamente comentado e quando estava, apresentava por vezes inúmeros erros. Foram também encontrados muitos blocos, que não se encontravam devidamente identificados, de código inacabado e outros que serviam para realizar alguns testes de *debugging*. Foi por isso necessário filtrar todo o código e separar o código propriamente dito, do restante, ao mesmo tempo que se corrigiam os vários erros encontrados para que todos os módulos ficassem a funcionar correctamente.

Ao mesmo tempo que se verificava todo o código, foi também necessário confirmar todas as ligações do circuito de cada módulo, corrigindo os maus contactos e alguns curto-circuitos detectados. Para tal, tínhamos apenas para nos guiar, o relatório do trabalho realizado no ano anterior, onde constavam apenas poucos esquemáticos de algumas ligações. Foi precisamente para confirmar as ligações que não constavam no relatório que esta análise teve de ser efectuada em conjunto com a análise do código, pois só assim conseguimos compreender a sua funcionalidade.

Todo o código foi ainda optimizado manualmente e correctamente comentado, a fim de facilitar futuras referências e alterações.

A análise, compreensão, correcção, organização e optimização do código de todos os módulos, correspondente à primeira tarefa dos objectivos propostos, demorou assim alguns meses, acabando por se prolongar mais do que inicialmente previsto. Este atraso deveu-se essencialmente ao facto deste não se encontrar mínima e correctamente comentado, com inúmeros erros de programação e extremamente desorganizado.

Depois de concluída a primeira tarefa, houve a necessidade de alterar e completar o código para que todos os módulos conseguissem comunicar conjuntamente. Este passo refere-se à segunda tarefa dos objectivos e foi também ele concluído com sucesso. A comunicação entre a buzina e a pista de luzes está concluída e a funcionar correctamente e o código final das luzes ocupam menos de 1*kB* de modo a caber no *MSP430F1101A*. A comunicação *wireless* ficou também ela concluída com um alcance máximo de aproximadamente 350*m* para os módulos de 433*MHz*. Todo o sistema de *firmware updates* ficou também ele a funcionar correctamente por *BSL*.

Não nos foi no entanto possível realizar a terceira tarefa dos objectivos propostos, tendo toda esta ficado para trabalho futuro.

7.2 - Trabalho Futuro

Não tendo conseguido concluir completamente este projecto, iremos realçar de seguida todos os objectivos que ficaram por concluir, assim como algumas soluções alternativas a ser exploradas para melhorar o seu funcionamento.

7.2.1 - *RTC*

A precisão do *RTC* da buzina deverá ser melhorada, de modo a que esta tenha um erro máximo de 100μs por hora. Esta precisão poderá ser obtida por *software*, com a alteração do valor do contador presente no *timer*, com a alteração da resolução do mesmo, com ambas as opções anteriores, com a criação de uma função específica que corrija este erro alterando o valor presente na variável *ctime*, ou ainda com a alteração do modo de contagem. Os modos de contagem podem ser *up*, *continuous* ou *up/down*. Deverão ser realizados todos os testes necessários para confirmar esta precisão.

7.2.2 - Simulador de *PDA*

Dever-se-á começar por rever as conexões existentes entre o *MAX232* e a porta série da *PDA* para verificar o seu correcto funcionamento de modo a concluir a ligação e a interface de comunicação

entre a *PDA* e o simulador de *PDA*. Inicialmente, a comunicação será testada com o envio de alguns *bytes* por parte da *PDA* e o simulador deverá receber todos estes dados sem dificuldades ou falhas. Quando este ponto estiver concluído, o *firmware* do simulador terá que ser alterado de modo a que este envie os dados recebidos pela *PDA* para a buzina, via *RF*. De seguida, será necessário testar o envio, por parte da *PDA*, de todos os comandos existentes, a sua recepção e transmissão, por parte do simulador, e a sua recepção pela buzina. Quando este ponto estiver concluído, deverá ser implementado na *PDA* o *firmware* da buzina (que tem incluído também o *firmware* das luzes) e este deverá ser enviado via *RS232* para o simulador que o transmitirá, via *RF*, para a buzina. Finalmente deverá ser eliminado, no *firmware* do simulador, todo o código existente, excepto o responsável pela inicialização e configuração do módulo *wireless* e o responsável pela recepção de comandos enviados pela *PDA* e posterior emissão dos mesmos para a buzina. Ficamos assim com o código final do simulador de *PDA*, que terá como únicos objectivos iniciar e configurar o módulo *wireless* para enviar e receber informação da buzina e da *PDA*.

7.2.3 - *PDA*

O código presente na *PDA* deverá ser todo ele revisto, optimizado e devidamente alterado para permitir o correcto funcionamento de todas as novas implementações, nomeadamente a comunicação *RS232* com o simulador de *PDA*, e outras que serão referidas de seguida.

7.2.4 - Detecção de duas luzes avariadas consecutivas

Como já foi referido no capítulo da pista de luzes, se existirem duas, ou mais, luzes avariadas consecutivas, poderão surgir graves problemas de funcionamento na mesma. Para evitar esta situação, foi desenvolvido um método que tenta detectar duas luzes avariadas consecutivas. Este método foi testado com sucesso numa placa branca, mas será agora necessário testar na própria pista de luzes.

Será necessário alterar o código da buzina para que esta, ao detectar uma luz avariada na pista de luzes, envie um comando para que o microcontrolador à esquerda do(s) avariado(s) configure o pino *I/O_R* como *output high*. De seguida, enviará um comando que configure os pino *I/O_L* e *I/O* do microcontrolador à direita do(s) avariado(s) como *output low*. Por fim espera um determinado intervalo de tempo e pede o resultado da comparação à pista de luzes.

Será ainda necessário alterar o *firmware* das luzes para que este reconheça os comandos enviados pela buzina e saiba que pinos e registos deverá configurar para efectuar correctamente a comparação.

No final deverá ser estudada ainda a hipótese de detectar avaria nos dois primeiros microcontroladores da pista de luzes, tal como foi efectuado ao longo desta dissertação para a detecção de avaria no microcontrolador da primeira luz.

7.2.5 - *MOSFET* de alimentação da pista de luzes

Embora o *MOSFET* que controla a alimentação da pista de luzes esteja a funcionar correctamente, foi detectada uma queda de tensão muito grande no dreno proporcional ao número de microcontroladores presentes na pista de luzes. Depois de efectuados vários testes, concluiu-se que este problema deve-se sobretudo à elevada resistência presente entre a fonte e o dreno (R_{DS} ou R_{on}). Para solucionar este problema deverão ser analisados outros *MOSFET* que apresentem uma resistência inferior e/ou colocar vários MOSFET em paralelo para reduzir essa mesma resistência. Esta última solução foi testada com sucesso, com a utilização em paralelo de dois *MOSFET BS250KL* e medimos uma redução para sensivelmente metade na queda de tensão no dreno.

7.2.6 - *Firmware Updates* na buzina

De momento apenas é possível efectuar *firmware updates* na buzina se o primeiro pacote recebido, quando se liga o sistema, for o pacote responsável pela sinalização de início do envio do novo *firmware* da buzina. Como o reprogramador da buzina entra em *LPM* assim que detecta um pacote que não o referido, e como existem apenas dois pinos que poderão ser controlados pela buzina (*P3.4* e *P3.5*) que não possuem interrupções, deverá ser estudado uma forma do reprogramador poder acordar ao fim de um determinado tempo para ler o estado de cada um desses pinos. Se a buzina detectar um pacote de sinalização de início do envio de novo *firmware*, ela altera o estado de um desses pinos e ficará à espera que o reprogramador saia de *LPM*. Quando o reprogramador sair de *LPM*, lê o estado do pino e inicia a sequência de entrada em *BSL* da buzina e configura o módulo *wireless* para receber os pacotes com o respectivo *firmware*. Se a buzina não detectar um pacote de sinalização de início do envio de novo *firmware*, ela não altera o estado do pino e quando o reprogramador sair de *LPM*, lê o pino e entra novamente em *LPM*.

7.2.7 - Outras soluções para o módulo wireless

De futuro deverão ser estudadas outras soluções para o módulo *wireless*. Entre os módulos a considerar, deve ser prestada grande atenção aos módulos *XBee*, *ZigBee* e *Bluetooth*. Deverão ser efectuados testes de consumo e de alcance para os módulos que se acharem boas alternativas para o *CC1000*.

7.2.8 - Construção de um protótipo

Depois será necessário concluir a última tarefa dos objectivos propostos que passa essencialmente pela construção de um protótipo final com o comprimento real de 25*m* de pista e deverão ser efectuados testes de integridade e testes de estanquicidade a fim de comprovar o correcto funcionamento de todos os módulos.

7.2.9 - Construção de um segundo protótipo

Para as piscinas de 50*m*, serão utilizadas 2 pistas de luzes com as suas respectivas buzinas instaladas em cada topo da piscina. Assim sendo, será necessário construir um segundo protótipo quando todos os testes forem correctamente efectuados no primeiro. Deverá ser implementado um algoritmo nas buzinas e na *PDA* para que esta atribua um *ID* diferente a cada uma das buzinas. Deste modo, será possível enviar comandos distintos para cada uma das buzinas, onde o *ID* de cada uma deverá ser incluído, por exemplo, no campo do comando de cada pacote. Deverão ainda ser implementados novos pacotes para estabelecer sincronismo entre as duas buzinas que deverão ser enviados regularmente para evitar desfasamentos temporais muito grandes.

7.2.10 - *Touchpad*

Deverá ainda ser estudado a implementação de um sensor, activado pelo toque do atleta, a ser colocado nos topos da piscina e que estará ligado directamente à *RTC* da buzina, a fim de registar com maior fiabilidade e precisão os tempos de volta e totais de cada atleta e o tempo de reacção após a sinalização de partida. Desta forma, poder-se-á obter uma base de dados para cada nadador com os seus melhores tempos e ter assim uma ideia mais global da sua evolução nos treinos. Como estes tempos serão completamente controlados pelos *touchpads* e pela *RTC* da buzina (sem intervenção do treinador), estes serão muito precisos (desde que a *RTC* tenha uma boa precisão como referido anteriormente).

7.2.11 - Adaptação a outras modalidades

Dever-se-á estudar ainda qual o melhor método a utilizar para adaptar este pacer para o treino de outras modalidades desportivas, nomeadamente o atletismo e ciclismo de pista e o remo.

Capítulo 8. Bibliografia

[Anacom, 2008] ANACOM, Autoridade Nacional de Comunicações – " Quadro Nacional de Atribuição de Frequências". Edição V1. Portugal. (2008) [On-line] Disponível em: http:// www.anacom.pt/streaming/qnaf2008_v1.pdf?contentId=955204&field=ATTACHED_FILE

[Barbosa, 2003] BARBOSA, T.; QUEIRÓS, T. – "A problemática da respiração no ensino da natação". Bragança: Escola Superior de Educação do Instituto Politécnico de Bragança. (2003) [On-line] Disponível em: http://www.efdeportes.com/efd58/natacao.htm.

[Cabrita, 2007] CABRITA, Hélder - "Pacer2 – Cadenciador de nova geração para desporto individual de alta competição - Hardware", Relatório final de conclusão da Licenciatura em Eng.ª Electrónica e Telecomunicações da Universidade de Aveiro. (2007).

[Dawley, 1985] DAWLEY, Dale K. – "Swimmer's lap pacer". United States Patent. Estados Unidos. (1985).

[Electronic Assembly, 2006] ELECTRONIC ASSEMBLY – "EA DIP 204-6". Alemanha. (2006) [On-line] Disponível em: http://www.lcd-module.com/deu/pdf/doma/dip204-6.pdf.

[Garber, 2008] - GARBER, Jarrod – "Pacer for athletes". World International Property Organization. África do Sul. (2006) p. 1.

[Marinho, 2003] MARINHO, D.; FERNANDES, R. – "A posição corporal nas técnicas alternadas em natação pura desportiva". Porto: Faculdade de Ciências do Desporto e de Educação Física da

Universidade do Porto. (2003) [On-line]. Disponível em: http://www.efdeportes.com/efd63/natacao.htm.

[Maxim, 2006] MAXIM – "+5V-Powered, Multichannel RS-232 Drivers/Receivers". Estados Unidos da América. (2006) [On-line] Disponível em: http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf.

[Piclist, 2009] PICLIST – "Ensuring byte-alignment for ASYNC over RF". (2009) [On-line] Disponível em: http://www.piclist.com/tecHREF/microchip/ammermansync.htm.

[Propox, 2009] PROPOX - "MMcc1000 User's Guide", Rev. 1. Polónia. (2009) [On-line] Disponível em: www.propox.com/download/docs/MMcc1000_en.pdf

[Sacadura, 1988] SACADURA, J.; RAPOSO, V. – "Metodologia do ensino das técnicas de nadar, partir e virar". Edição Ministério da Educação – Desporto e Sociedade. (1988).

[Silva, Ricardo] SILVA, Ricardo – "Projecto e implementação do software da interface gráfica no cadenciaddor", Relatório final de conclusão da Licenciatura em Eng.ª Electrónica e Telecomunicações da Universidade de Aveiro. (2007).

[STMicroelectronics, 2009] STMICROELECTRONICS – "LD1117xx Low drop fixed and adjustable positive voltage regulators". Suiça. (2009) [On-line] Disponível em: www.st.com/stonline/products/literature/ds/2572.pdf.

[Termin, 1999] TERMIN, Budd; [et al.] – "Pace lights and swim performance". Swimming Technique 36 (Out.-Dec. 1999) p. 18-20.

[Texas Instruments, 1999] TEXAS INSTRUMENTS – "MSP430C11x1, MSP430F11x1A, Mixed signal microcontroller". Estados Unidos da América. (1999), revisto em Dezembro de 2008 [On-line] Disponível em: http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=msp430c1101&fileType=pdf.

[Texas Instruments, 2001] TEXAS INSTRUMENTS – "Features of the MSP430 Boostrap Loader". Estados Unidos da América. (2001) [On-line] Disponível em: http://trac.edgewall.org/raw-attachment/ticket/1921/slaa089a.pdf.

[Texas Instruments, 2002] TEXAS INSTRUMENTS – "MSP430x15x, MSP430x16x, MSP430x161x, Mixed signal microcontroller". Estados Unidos da América. (2002), revisto em Maio de 2009 [On-line] Disponível em: http://focus.ti.com/lit/ds/symlink/**msp430f1611**.pdf.

[Texas Instruments, 2009] TEXAS INSTRUMENTS – "CC1000 Single Chip Very Low Power RF Transceiver". Estados Unidos da América. (2009) [On-line] Disponível em: http://focus.ti.com/lit/ds/symlink/cc1000.pdf